Departamento de Informática

PROYECTO FIN DE CARRERA

Diseño e implementación de una aplicación Android para realizar presentaciones

Autor: Juan Manuel Oviedo Expósito

Tutor: Alejandro Calderón Mateos

Leganés, 29 de Octubre de 2010

Agradecimientos

Agradezco a mis padres, José Manuel y Encarnación, por haberme dado una educación excepcional dentro de sus posibilidades pero sobre todo por haberme enseñado dos lecciones muy importantes: decir lo que uno piensa pero sobre todo saber cómo decirlo.

A mi hermana Carolina, en primer lugar poder decir que tengo una gran amiga en ella, y en segundo lugar todo el ánimo que me ha dado siempre a lo largo de la carrera.

A mi tutor de este proyecto, Alejandro Calderón, por ser un magnifico profesor y haberme dado tan buenos consejos.

A mis amigos Soraya y Óscar por estar ahí siempre sin condiciones. A Jesús y Patricia por todas las historietas vividas desde el "castillo" de Camelot en adelante. A Francisco José (alias Kiko) y Natalia porque por mantener nuestra amistad estoy recorriendo toda la geografía española y por tener dos niñas que son un sol.

A Laura y Óscar por tantas horas a lo largo de estos 7 años en los que aparte de currar mucho sobre todo nos hemos reído, ya fuese comiendo en la cafetería o haciendo una práctica.

A Lidia y Roberto por los viajes en coche, las discusiones interminables… pero sobre todo por la risa "maligna" de Lidia y la naturalidad de Roberto.

A Álvaro por ser mi último mono y tener siempre una buena conversación. A Christian por estar siempre dispuesto a ayudar. A Iván por tener un inigualable sentido del humor. A Javi Roca por tener tanta paciencia. Al laboratorio de Informática (Jaime, Alberto, Óscar) por los desayunos en la cafetería del Sabatini que tanto se echan de menos.

A todos los "informatilocos" por la cantidad de momentos que hemos vivido juntos desde que comenzase la beca en Getafe.

En definitiva gracias a todos los que durante estos años, en algún momento hemos compartido una carcajada, pues al final es lo que queda, los buenos ratos.

Por último quería recordar con un especial cariño a mis abuelas, Manuela y Victoria, a las que me hubiera gustado decir: "¡He terminado!".

Resumen

En este documento trata de ofrecer una panorámica general acerca del desarrollo de aplicaciones para dispositivos que dispongan del sistema operativo Android de Google. Para ello se estudiarán las características técnicas de la plataforma mencionada analizando las ventajas e inconvenientes respecto a otros entornos de desarrollo disponibles. Además, se incluirá un caso práctico de desarrollo de una aplicación para la plataforma que abarca las fases de análisis, diseño e implementación dentro del ciclo de vida del software.

Palabras clave: Android, aplicación, análisis, diseño, implementación

Abstract

This document intends to show a general view about the development of applications for any kind of device that runs the Operating System Google's Android. Due to this, the technical features of the mentioned platform will be studied and it will be analyzed the advantages and disadvantages in regard to other available platforms for development. Furthermore, the development of an application will be included within the document that will include the analysis, design and implementation phases of software lifecycle.

Keywords: Android, application, analysis, design, implementation

Índice General

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

El proyecto desarrollado y del que se hablará en el presente documento surge a partir de una motivación principal que es poder controlar un dispositivo de forma remota utilizando para ello un teléfono móvil.

Bajo esta premisa se ha desarrollado una aplicación para la plataforma Android que permite, entre otras cosas, interactuar de forma remota con otro dispositivo, como por ejemplo un ordenador personal que esté preparado para ello, pudiendo controlar una presentación de diapositivas o de imágenes realizando gestos en la pantalla del teléfono móvil.

La aplicación por tanto tiene un claro objetivo docente pues con ella se pretende solucionar el problema de transportar los documentos, presentaciones o imágenes que se pueden utilizar para impartir una clase usando un teléfono móvil para ello, de forma que a través de una conexión inalámbrica se puedan transferir los archivos a un equipo y manipularlos de forma remota.

1.2 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación que ayude al usuario a realizar presentaciones, a la vez que se puede gestionar los contenidos del teléfono móvil de forma cómoda y sencilla, utilizando para ello la interfaz de la pantalla táctil con la que cuentan estos dispositivos.

Como consecuencia del desarrollo de dicha aplicación, los objetivos que busca conseguir este proyecto son los siguientes:

- Ofrecer una panorámica de las distintas plataformas móviles que existen en el mercado en la actualidad analizando las posibilidades que ofrece cada una para un desarrollador de software
- Conocer los detalles de la arquitectura de Android, así como obtener los conocimientos necesarios para desarrollar aplicaciones para ella
- Establecer los pasos necesarios para desarrollar aplicaciones abarcando desde la creación del proyecto hasta la publicación del mismo

1.3 Fases del desarrollo

En la primera fase del desarrollo se ha llevado a cabo una investigación acerca de la plataforma Android, estudiando las principales características de la misma como la arquitectura, librerías, framework de desarrollo, etc. Una vez terminado el estudio de la plataforma se pusieron en práctica los conocimientos adquiridos acerca de la misma, mediante la realización de distintos tutoriales sencillos.

En la siguiente fase, se evaluó el desarrollo de distintas aplicaciones para Android. Una vez quedó definida la aplicación que se quería desarrollar, se estudio si la plataforma ofrecía los medios necesarios para llevar a cabo dicha aplicación.

Tras comprobar que Android satisfacía los requisitos necesarios para la aplicación pensada, en la siguiente fase del proyecto empezó el desarrollo de la misma llevándose a cabo el diseño e implementación.

En la última fase del desarrollo se han llevado a cabo numerosas pruebas para comprobar el comportamiento correcto de la aplicación, corrigiendo los errores encontrados y optimizando el funcionamiento de la misma.

1.4 Medios empleados

Para la elaboración de este proyecto se ha contado con un equipo de desarrollo con las siguientes características:

- Sistemas operativos: Windows 7 Profesional 64 bit y Ubuntu 10.04 LTS 64 bit
- Librerías de desarrollo: Java SDK6 y Android SDK R07
- Herramientas de desarrollo: IDE Eclipse 3.6 Helios con el complemento ADT v0.98 instalado

Dado que el proyecto consiste en una aplicación de tipo cliente-servidor, la parte servidora del mismo se ha suplido con el mismo equipo de desarrollo. Así mismo, para la depuración y pruebas de la aplicación cliente se ha utilizado un teléfono móvil HTC Magic con Android 2.2 instalado en él, además del emulador con el que cuenta el complemento ADT para Eclipse.

Por último, para la creación de la documentación del proyecto se ha utilizado la suite ofimática Microsoft Office 2007 Profesional.

1.5 Estructura de la memoria

En este apartado se comentará de forma breve el contenido de los capítulos del presente documento con el objetivo de facilitar la lectura del mismo:

- Capítulo 1: breve introducción en la que se exponen los objetivos y motivaciones del proyecto, las fases de desarrollo, o los medios empleados para llevarlo a cabo
- Capítulo 2: comprende un análisis del estado actual de mercado de la telefonía móvil desde el punto de vista del desarrollo de software, haciendo hincapié en las posibilidades que ofrecen las distintas plataformas actuales
- Capítulo 3: consiste en un resumen de las características principales de la plataforma como son la arquitectura, librerías, componentes de una aplicación, etc. La información contenida en este capítulo se puede encontrar mucho más detallada en inglés en la web de desarrollo de Android [13]
- Capítulo 4: la intención del mismo es demostrar con un sencillo ejemplo práctico como crear una aplicación siguiendo una serie de pasos que comprenden desde la creación del proyecto en Eclipse hasta la publicación del mismo

- Capítulo 5: este capítulo comprende la fase de análisis del ciclo de desarrollo de software y se explicarán aspectos del diseño y la implementación
- Capítulo 6: en él se explicarán las conclusiones obtenidas al finalizar el desarrollo así como se indicarán una serie de líneas futuras que podrían ser aplicadas al proyecto
- Capítulo 7: el último capítulo de este documento contendrá la planificación seguid apara el desarrollo del proyecto y un presupuesto del coste real de la aplicación en términos económicos
- Glosario: al final del documento se incluye un glosario de los términos utilizados en este documento que ayuden a la mejor comprensión del mismo por parte del lector
- Referencias y Bibliografía: por último se incluyen las referencias y la bibliografía consultada para la elaboración del proyecto.

Capítulo 2

Estado de la cuestión

2.1 Introducción

Hoy en día hablar del término dispositivo móvil puede dar lugar a una gran variedad de interpretaciones distintas, ya que existen multitud de aparatos electrónicos que poseen muchas de las características que se le presuponen a un dispositivo de este tipo.

Si tenemos en cuenta una definición más o menos rigurosa de este término como es la siguiente: [1] "*aparatos de pequeño tamaño, con algunas capacidades de procesamiento, móviles o no, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales*", podemos comprobar que como se mencionó anteriormente hay una gran variedad de aparatos que cumplen esta definición, en mayor o menor medida, como por ejemplo una PDA, un teléfono móvil, un lector de libros electrónico o un ordenador portátil.

De entre los elementos mencionados destaca, sin lugar a dudas, el teléfono móvil como el dispositivo más utilizado de entre todos. No obstante, en España la media de teléfonos móviles por persona es superior a un teléfono por habitante (como así lo indican los datos de la Comisión del Mercado de Telecomunicaciones a fecha de Marzo del 2009 [2]). Si vamos más allá de nuestras fronteras, y nos fijamos en los datos a nivel mundial del número de líneas móviles, comprobamos que hay estimado un total de 4.600 millones de líneas [3].

Teniendo presentes estos datos es lógico pensar que el mercado de la telefonía móvil resulta muy interesante desde un punto de vista económico o de oportunidad de negocio. Como ingeniero informático, una de estas oportunidades a las que hago referencia es desarrollar software para este tipo de dispositivos.

Figura 1: Distintos modelos de teléfono móvil

Es evidente que no todos los teléfonos móviles tienen las mismas características o permiten realizar las mismas funciones. Si bien hace unos años los teléfonos servían exclusivamente para hablar y mandar SMS, hoy en día un móvil te permite entre otras cosas reproducir contenidos multimedia, conectarte a internet desde cualquier punto, sacar fotografías, utilizarlo como dispositivo GPS, etc. La presencia de esta serie de características añadidas en un teléfono móvil ha hecho que se empiece a utilizar un término nuevo para referirnos a estos aparatos.

Figura 2: Algunos modelos de Smartphone

Nace así el concepto de "*smartphone*"[4] o en castellano "*teléfono inteligente*" para denominar a los teléfonos móviles que ofrecen estas capacidades (acceso al correo, internet, etc.) y que además suelen presentar otros atributos físicos como son la presencia de pantallas táctiles que permiten manejar el teléfono o la incorporación de teclados QWERTY [19] como dispositivos de entrada de texto.

Todas estas capacidades o funciones mencionadas anteriormente estarían desaprovechadas sin software que las saque partido. Es por ello que en la actualidad oímos hablar de las tiendas o servicios de venta en línea de aplicaciones para teléfonos móviles como el "*App Store*" de Apple [20] o el "*Android Market*" de Google [21]. Estos servicios se caracterizan por ofrecer al usuario un inmenso catálogo de aplicaciones, algunas gratuitas y otras de pago, que aportan un valor añadido al dispositivo pues van a permitir que el usuario pueda abrir documentos de texto, ver presentaciones, editar sus fotografías o consultar un mapa para orientarse.

Con el panorama descrito, en el que cada vez más a menudo, dada la tendencia actual del mercado, los teléfonos que aparecen son pequeños ordenadores personales a los que hay que nutrir de software que aprovechen sus capacidades; como posibles desarrolladores nos surgen las siguientes preguntas: "*¿Qué puedo ofrecer?"*, algo que ya fue comentado en el primer capítulo, y otra pregunta, que como se verá en el siguiente apartado, resulta ser un factor clave "*¿Dónde debo ofrecerlo?*".

2.2 Estudio de las plataformas móviles actuales

En este apartado se discutirán los principales aspectos de las plataformas de desarrollo para smartphones actuales: iOS 4, Symbian^3, Windows Phone 7 y Android 2.2, analizando sus ventajas e inconvenientes desde el punto de vista del desarrollo de software y relacionándolo con el del usuario.

En la siguiente tabla se comparan distintos aspectos de los plataformas mencionadas [8], [9]:

CARACTERISTICA	PLATAFORMA			
	Android 2.2	**iOS 4.0**	**Symbian^3**	**Windows Phone 7**
Buscador integrado	Si	Si	Si	Si
Compartir Internet con otros dispositivos	Si – USB, hotspot Wifi o Bluetooth	No – Requiere modificar el dispositivo	Si - Wifi	N/D
Flash 10.1	Si	No	No – versión Lite	No
Gestor de correo	Si	Si	Si	Si
Inicio personalizable	Si	Si – Limitado	Si	Si
Integración con redes sociales	Si	Si	Si	Si
Juegos	Si – Market	Si – App Store	Si	Si
Libros (eBooks)	Si	Si	Si	No
Mapas con navegación GPS (gratuita)	Si – Google Maps	No	No	Si – Bing Maps
Multi-tarea	Si	Si – Limitado a ciertas aplicaciones y servicios	Si	Si – Limitado a ciertas aplicaciones y servicios
Multi táctil	Si	Si	Si	Si
Navegador Web	Si	Si	Si	Si
Servicios en "la nube"	Si	Si	Si	Si
Silverligth	No	No	No	Si
Suite Ofimática	Si	Si	Si	Si
Teclado	Físico y virtual	Solo virtual	Físico y virtual	Físico y virtual

Temas personalizables para la IU	Si	Si – Limitado	No	No
Tienda de música	Si – Terceras partes	Si – iTunes	Si – Terceras partes	Si – Zune
Tienda de aplicaciones	Si	Si	Si	Si

Tabla 1. Comparativa plataformas

Analizando los datos de la anterior tabla, la primera impresión es que sobre el papel las cuatro plataformas no difieren mucho en sus principales características, si bien hay ciertos detalles que las hacen más atractivas a unas sobre otras desde el punto de vista de usuario y también desde el punto de vista de desarrollador.

Por un lado, desde la perspectiva del usuario habitual de la telefonía móvil, estos se suelen centrar en aspectos como la capacidad de personalización de la interfaz de usuario o lo bien que ve esta, si tienen un montón de aplicaciones gratuitas, o si la plataforma cuenta con un amplio catálogo de juegos.

Teniendo en cuenta lo mencionados en el párrafo anterior, es fácil darse cuenta que las plataformas que cumplen en mayor grado estas condiciones son Android e iOS.

Respecto a la capacidad de personalización del sistema, la plataforma que permite un mayor grado de libertad a sus usuarios es Android. Prácticamente cualquier elemento puede ser configurado por el usuario, desde un simple fondo de escritorio hasta la interfaz de usuario completa aplicándole algún tema visual.

Figura 3. Interfaz de usuario en las diferentes plataformas (Symbian^3, Android 2.2, iOS 4 y Windows Phone 7)

Por otro lado, Apple en este sentido es más restrictivo, pues todos los dispositivos en los que IOS está presente comparten la misma interfaz de usuario, permitiendo muy pocos cambios en la misma. Si bien es cierto que la interfaz diseñada para iPhone y demás dispositivos de Apple cuentan con una magnifica interfaz de usuario..

Respecto a la segunda cuestión mencionada con anterioridad, es decir, las aplicaciones, actualmente ni Nokia con su OviStore ni Microsoft cuentan con una tienda de aplicaciones que pueda estar a la altura de sus otros dos rivales, hablando principalmente en términos de cantidad de aplicaciones, si bien la intención de Microsoft con Windows Phone 7 es introducir una nueva tienda cuando este sistema llegue a los mercados a finales de Octubre de 2010.

Android Market y App Store cuentan con una gran cantidad de aplicaciones, gratuitas y de pago, y también con un extenso catálogo de juegos, de hecho, en la actualidad la App Store cuenta con más de 250.000 aplicaciones y juegos, y por su parte el Market de Android cuenta con cerca de 100.000. No obstante hay que mencionar que en ambas plataformas hay numerosas aplicaciones que no presentan un mínimo de calidad o tienen como objetivo el simple hecho de gastar una broma.

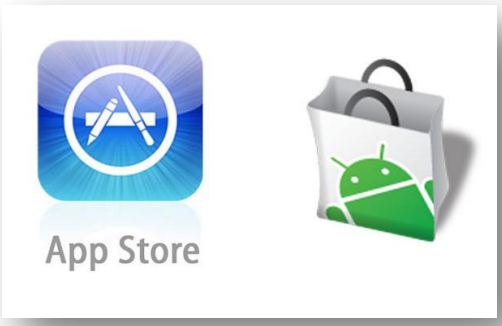

Figura 4. Tiendas de aplicaciones en iOS y Android

Cambiando el punto de vista al de un desarrollador de software, y relacionado con lo anterior, resulta evidente que en ambas plataformas existe una gran oportunidad de negocio, pues cada una de ellas cuenta con millones de usuarios. Por tanto la siguiente pregunta a resolver es "*¿Cuál de las dos?*". Para ello se van a discutir varios aspectos como la cuota de mercado, las facilidades de desarrollo, etc.

Respecto a la cuota de mercado, en el siguiente gráfico, realizado a partir de un estudio de ComScore [11][15], se puede apreciar la cuota estimada para cada una de las plataformas en Estados Unidos a Julio de 2010:

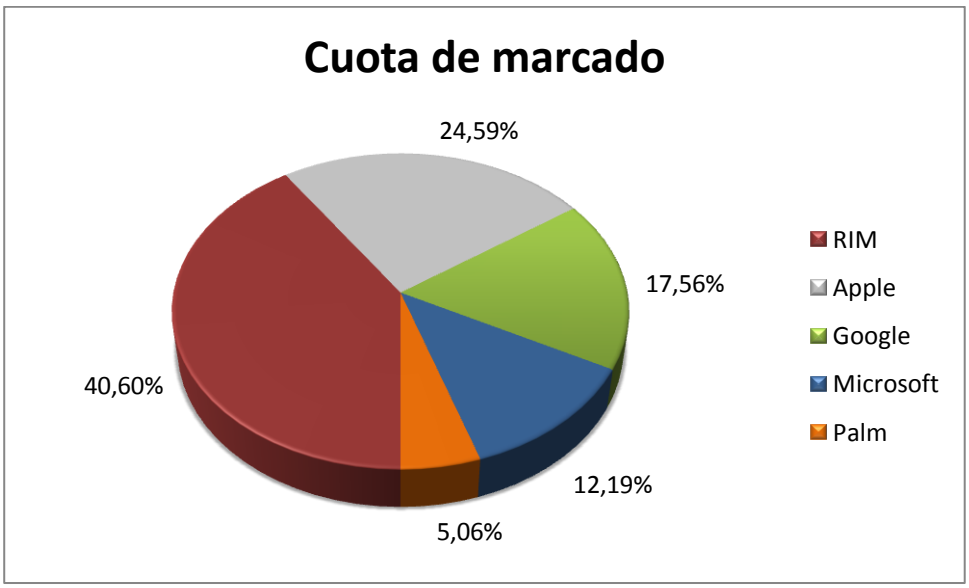

Figura 5. Cuota de mercado smartphones EUA a Julio de 2010

Analizando estos datos, se puede ver que en el mercado norteamericano cuenta con una mayor cuota de mercado RIM con sus Blackberry (aunque también es cierto que esta plataforma y sus terminales están orientados al mundo empresarial y cuentan con una tienda de aplicaciones bastante más reducida), seguida de Apple y Google. En otros mercados se da una situación bastante parecida, como por ejemplo en Europa, donde es Nokia con Symbian la marca dominante.

Sin embargo, según indican algunos estudios como el realizado por Gartner Forecast [10], esta situación dará un cambio significativo alrededor de 2014, como se puede ver en la siguiente figura, donde se muestra la evolución de las plataformas desde 2009 a 2014, y en la que claramente la plataforma con un mayor crecimiento esperado es Android:

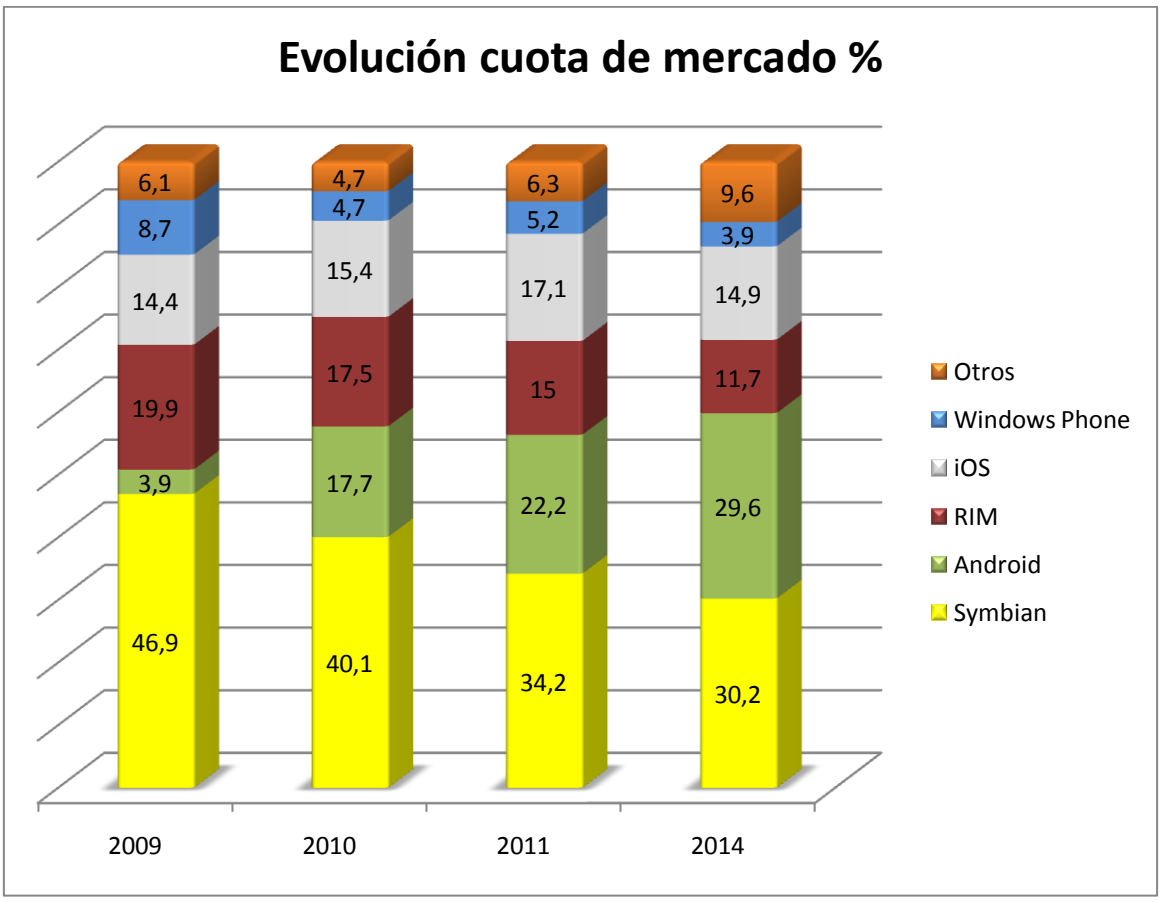

Figura 6. Estimación cuota de mercado por plataforma para 2014

Con estos datos se refuerza aún más la idea inicial de realizar el proyecto en una de las dos plataformas mencionadas, Android e iOS; pues ambas están pensadas exclusivamente para el mercado de smartphones (o dispositivos similares tipo iPad) y son las que mayor cuota de mercado tienen previsto alcanzar a corto plazo en detrimento de otras como Symbian.

A parte de todo lo mencionado anteriormente se deben tener en cuenta otras consideraciones para determinar la plataforma elegida y que están relacionadas con el propio proceso de desarrollo de aplicaciones para las mismas.

Ambas plataformas, iOS y Android, cuentan con bastantes herramientas de desarrollo, extensas APIs, multitud de librerías que extienden las funcionalidades de la plataforma y numeroso código fuente de ejemplo.

Sin embargo, desde mi punto de vista, Android cuenta con ciertas ventajas competitivas como poder desarrollar de forma independiente a la plataforma ya que en ningún caso requiere de hardware específico para el desarrollo, por el contrario iOS requiere contar con un equipo que tenga instalado Mac OS X; o el coste de la licencia de desarrollador siendo más baja para Android.

Además, Apple ha establecido una política de privacidad bastante restrictiva, en la que hasta hace poco no se permitía a los desarrolladores hablar sobre el SDK o compartir código, sin mencionar que se deben obtener varios certificados emitidos por la compañía de la manzana incluso para probar las aplicaciones desarrolladas en nuestro iPhone. Por el contrario, Google con Android adopta una política más abierta facilitando mucho más el desarrollo de aplicaciones en lo relativo a estos aspectos.

Por último, otras consideraciones relativas al desarrollo, como el lenguaje de programación para cada plataforma dependerán de la experiencia de cada persona, y en mi caso particular, he programado en Java en muchas ocasiones.

En conclusión, y en base a todo lo expuesto, se ha escogido Android para llevar a cabo este proyecto por las siguientes razones:

- ✓ Oportunidad de negocio a corto/medio plazo
- ✓ Política de desarrollo menos restrictiva
- ✓ Inferiores costes de desarrollo (equipamiento y licencia)
- ✓ Lenguaje de programación y herramientas

2.3 Comparativa con otras soluciones

Una vez elegida la plataforma de desarrollo de la aplicación, se va a proceder a realizar un estudio de viabilidad del proyecto, realizando una búsqueda de alternativas y comprando la solución que se quiere desarrollar con otras que ya existan en el mercado.

Dentro de Android Market se han encontrado varias aplicaciones similares a la que se quiere desarrollar, de entre ellas se han elegido dos por tener una buena valoración por parte del usuario y un número considerable de descargas: GMote [17] y PPT Remote [18].

En la siguiente tabla se van a comparar distintas características de las propuestas para la aplicación que se quiere desarrollar con las características de las aplicaciones mencionadas:

CARACTERÍSTICA	UC3M Slides	GMote	PPT Remote
Multiplataforma	✓	✓	✗ Solo Windows
Explorador de archivos integrado (permitiendo copiar, pegar, etc.)	✓	✗	✗
Envío y recepción de archivos	✓	✗	✗
Comunicación redes locales e Internet	✓	✗ Solo red local	✗ Solo red local
Wifi / 3G / Bluetooth	✓ / ✓ / ✗	✓ / ✗ / ✓	✓ / ✗ / ✓
Controlar PowerPoint (1 click)	✓	✓	✓
Controlar PDF (1 click)	✓	✗	✗
Controlar presentaciones imágenes (1 click)	✓	✗	✗
Captura de pantalla remota	✓	✗	✗
Modo “Mouse”	✗	✓	✓
Restricciones de uso	✗	✗	✓ 10 diapositivas versión gratuita

Tabla 2. Comparativa aplicaciones

Como se puede comprobar la aplicación desarrollada en este proyecto presenta más funcionalidades que las otras dos. Las únicas características que no se han implementado para esta versión, y que estas aplicaciones poseen, son la comunicación por bluetooth y el control del ratón de forma remota. Sin embargo, la aplicación *UC3M Slides* permite gestionar los archivos locales del teléfono, enviar y recibir archivos, comunicación 3G, acceso desde redes públicas o capacidad para crear anotaciones de presentaciones llevadas a cabo por otra persona.

Capítulo 3

Android en detalle

3.1 Introducción

Android es una plataforma de software [5], pensada especialmente para dispositivos móviles, y que en pocas palabras se compone de un sistema operativo, un conjunto de aplicaciones base y una capa *middleware* situada entre las aplicaciones y el propio sistema operativo.

Dichas aplicaciones base están desarrolladas utilizando el lenguaje de programación Java, mientras que por otro lado ciertos componentes de la arquitectura de Android como las librerías o el núcleo, basado en el *kernel* de Linux 2.6; están escritos en C/C++.

En los siguientes apartados serán descritos distintos aspectos de la plataforma como su arquitectura, los componentes de una aplicación, etc., haciendo hincapié en los aspectos a priori más importantes. Toda esta información ha sido obtenida de la página oficial de desarrollo de Android [13], que ha sido traducida y resumida para incorporarla a este proyecto.

No obstante y de forma introductoria, a continuación serán expuestas las principales características de Android:

- Cuenta con un **framework de aplicaciones** que permite **reutilizar** o **sustituir** las aplicaciones existentes.

- Cuenta con una **máquina virtual** especialmente **optimizada para dispositivos móviles** conocida como Dalvik VM.
- **Navegador** integrado basado en el **motor** de código abierto **WebKit**.
- Capaz de procesar **gráficos en 2D** (gracias a la librería SGL) y **gráficos 3D** basados en la especificación OpenGL ES 1.X/2.0 (dependiendo de la versión de Android).
- Soporte para el almacenamiento de datos estructurados mediante las librerías **SQLite**.
- Soporte de **múltiples formatos multimedia** tanto de audio como video o imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- **Telefonía** GSM.
- **Comunicaciones** siguiendo los protocolos estándar **Bluetooth**, **EDGE**, **3G**, y **Wifi**.
- Soporte de múltiples dispositivos hardware como **cámaras**, **GPS**, **brújula** o **acelerómetros**.
- Potente entrono de desarrollo que incluye un emulador, herramientas de depuración o un complemento par el IDE Eclipse.

3.2 Evolución de la plataforma

En el apartado anterior se listaron las principales características de la plataforma, no obstante Android ha ido evolucionando de forma notable con el paso del tiempo, desde que se publicara su primera versión a finales del 2008, añadiendo mejoras y características nuevas. Actualmente, son tres las principales versiones de Android: la versión 1.6, de nombre en clave *Donut*, la 2.0.1/2.1 conocidas con el sobrenombre de *Eclair* y la reciente versión 2.2, bautizada con el nombre de *Froyo*.

Por supuesto también existieron versiones anteriores a las mencionadas: la 1.0, con la que se lanzó el primer móvil Android del mercado, el HTC Dream o G1, en Octubre de 2008; la 1.1, de febrero de 2009, que solucionaba varios errores y añadía alguna que otra funcionalidad no demasiado importante; y la que fue la primera gran actualización de la plataforma, la versión 1.5 con el alias de *Cupcake* lanzada en Abril de 2009.

En la tabla siguiente se pueden ver con detalle las principales mejoras aportadas por cada versión [6][7], así como un pequeño anticipo de la siguiente iteración de la plataforma conocida como *Gingerbread*/3.0:

Versión	Características
1.5	Debido a las grandes mejoras introducidas en la tercera versión de Android el número de versión saltó directamente a la 1.5 desde la 1.1. Basado en el *kernel* Linux 2.6.27, las novedades más interesantes eran las siguientes: ○ Rediseño completo de todos los elementos de la interfaz. ○ Transiciones animadas entre ventanas. ○ Mejoras en la velocidad de la cámara. ○ Menor tiempo de búsqueda de los satélites GPS, gracias a la posibilidad de utilizar A-GPS. ○ Mejoras en la velocidad del navegador web gracias a la inclusión de la última versión del motor de renderizado WebKit así como del intérprete de JavaScript SquirelFish. ○ Añadida la posibilidad de copiar y pegar texto, así como de poder buscar texto dentro de una página web. ○ Posibilidad de personalizar los elementos mostrados en el escritorio. ○ Inclusión de teclado en pantalla, con soporte para orientación vertical y apaisada, funcionalidades de auto corrección y soporte de diccionarios del usuario. ○ Añadida la posibilidad de grabar y reproducir vídeos. ○ Añadido soporte Bluetooth A2DP y AVRCP.
1.6	Lanzada en Septiembre de 2009, está basada en el *kernel* de Linux 2.6.29. Se considera una actualización menor, pero aun así se introducen algunas novedades interesantes: ○ *Quick Search Box*, motor de búsqueda en la pantalla de inicio que permite buscar entre distintas fuentes (contactos, historial del navegador, Google). Con autocompletado y capacidad de aprendizaje. ○ Mejorada la velocidad de la cámara. ○ Posibilidad de conectarse a redes VPN, 802.1x. ○ Nuevas opciones de configuración para controlar el uso de la

	batería, que permite comprobar qué aplicaciones y servicios son los que más consumen. Desde esta pantalla se puede también parar o desinstalar las aplicaciones instaladas en el dispositivo. o Las aplicaciones de *Android Market* aparecen ahora ordenadas por categorías (Aplicaciones, Juegos y Descargas). Para cada categoría podemos consultar las últimas actualizaciones y las aplicaciones más populares. Además para cada aplicación se muestran capturas de pantalla y opiniones de otros usuarios. o Nuevo motor de texto a voz. o Añadidas herramientas para la creación de gestos (patrones dibujados en pantalla para realizar tareas específicas como añadir un marcador al navegador o copiar y pegar texto).
2.0/ 2.0.1	En noviembre de 2009 se lanza la versión 2.0, continuando con la tradición de utilizar dulces de repostería como nombres para las versiones, las novedades más importantes de *Eclair* son: o Optimizaciones para mejorar el rendimiento de los dispositivos. o Mejoras en la interfaz de usuario. o Rediseñó de la interfaz del navegador, contando ahora con soporte para distintas características de HTML5 (entre ellas la etiqueta vídeo), la posibilidad de hacer zoom con una doble pulsación y miniaturas de los marcadores. o Soporte nativo de flash para la cámara. o Añadido zoom digital, modo escena, balance de blanco, efectos de color y modo macro. o Mejoras en el teclado virtual. o Soporte para nuevos tamaños y resoluciones de pantalla. o Rediseño de los contactos. o Bluetooth 2.1. o Soporte nativo de aplicaciones como Facebook. o Mejoras en Google Maps, que pasaba a ser multi-táctil y soportar capas de visionado.

	○ Soporte de Microsoft Exchange. ○ Mejoras en el calendario. En diciembre de 2009 se publicó una pequeña revisión, Android 2.0.1, que mejoraba la duración de la batería y la estabilidad, añadía la funcionalidad de llamada a tres, mejoras en el GPS, el Bluetooth, y la velocidad de disparo y auto foco de la cámara.
2.1	Android 2.1, es lanzado en enero de 2010, también se considera una actualización menor aunque incluye mejoras importantes: ○ Reconocimiento de voz pudiéndose dictar frases o palabras en lugar de escribirlas en cualquier campo de texto. ○ Mejoras en el teclado virtual. ○ Galería de imágenes 3D. ○ Añadidos nuevos gestos para hacer zoom en el navegador, la galería y en Google Maps. ○ Rediseño de aplicaciones e inclusión de algunas nuevas como la aplicación de tiempo y noticias o de realidad aumentada como Google Googles. ○ Mejoras en Google Maps como la sincronización de nuestros sitios favoritos, el modo noche y auto completado de búsquedas. ○ Mejoras en la duración de la batería.
2.2	Lanzado en Junio de 2010, *Froyo* cuentas con las siguientes características: ○ Actualizaciones automáticas para las aplicaciones. ○ Soporte Wifi 802.11n. ○ Soporte para Radio FM. ○ Soporte Flash 10.1 y Adobe AIR 2.5. ○ Soporte OpenGL 2.0. ○ Inclusión de un compilador JIT que mejora entre 2 y 5 veces el rendimiento de la versión 2.1. ○ Posibilidad de utilizar el dispositivo como modem utilizando la

	conexión USB así como de punto de acceso Wifi. o Incorporación del motor de JavaScript V8 del navegador Chrome. o Posibilidad de mover una aplicación instalada desde el teléfono a la tarjeta de memoria, y viceversa. o Opciones avanzadas de gestión energética. o Incluidas nuevas opciones de pago de aplicaciones como PayPal y Google Check Out.
3.0	Con un lanzamiento previsto para Octubre de 2010, de esta versión se conocen aún muy pocos detalles, aunque si se ha confirmado que requerirá de dispositivos de altas prestaciones (CPU a 1 GHz, 512 MB de RAM, pantallas de 3,5 pulgadas o superiores). Las mejoras que se espera que incluya son las siguientes: o Soporte de resoluciones de hasta 1366x768 pixeles (WXGA). o Interfaz de usuario renovada. o Soporte para reproducción de video en formato WebM. o Tienda de música similar a iTunes.

Tabla 3: Versiones de Android y sus características

3.3 Arquitectura

En la siguiente figura podemos ver un diagrama que muestra los principales componentes de la arquitectura de Android:

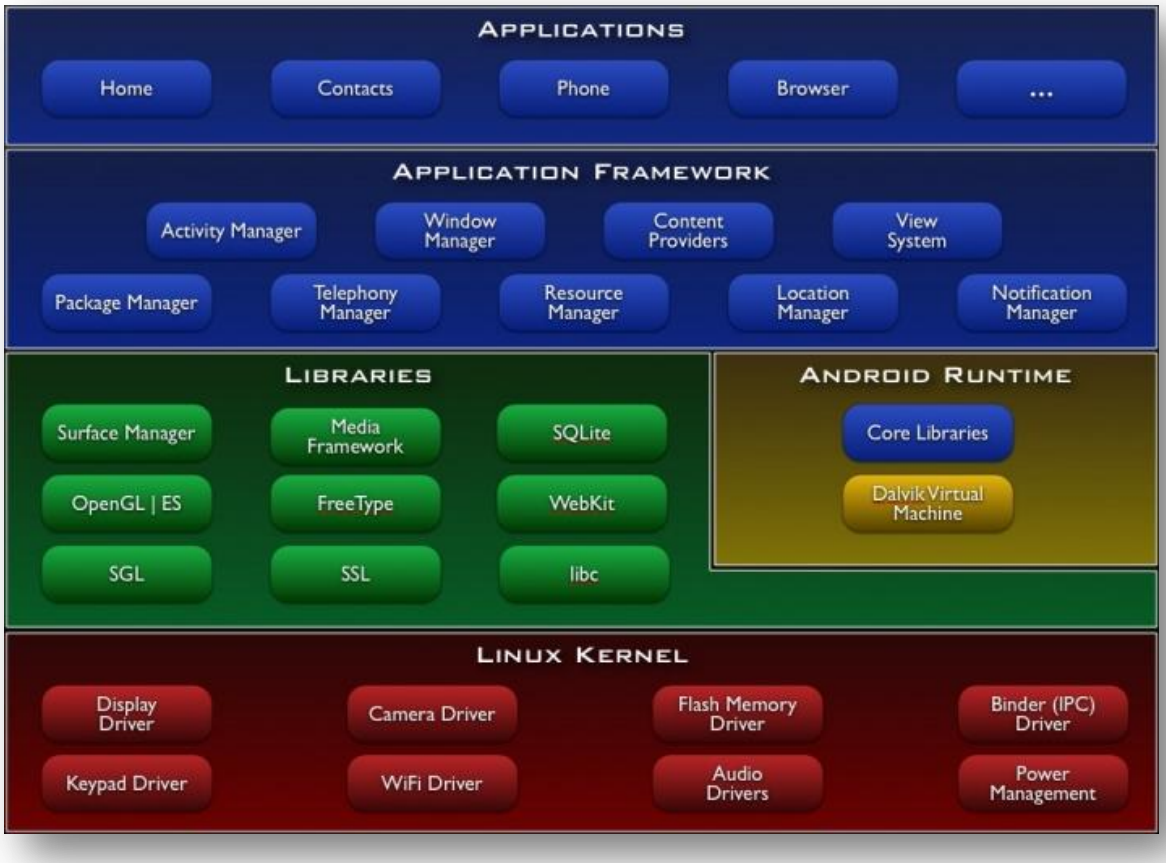

Figura 7: Arquitectura de Android

3.3.1 Nivel de aplicaciones

Android incluye una serie de aplicaciones básicas como un cliente de correo electrónico, un programa para mandar SMS, calendario, mapas, navegador web o contactos entre otros. Todas las aplicaciones están escritas en el lenguaje de programación Java y su desarrollo es posible gracias al Android SDK que provee de las herramientas e interfaces de programación necesarias.

3.3.2 Nivel del framework de aplicaciones:

Android ofrece a los desarrolladores la capacidad de utilizar cualquier dispositivo presente en el teléfono para desarrollar una aplicación, pudiendo acceder por ejemplo a información de geolocalización a través del GPS; ejecutar servicios en

segundo plano, establecer alarmas, añadir notificaciones a la barra de estado, entre otras muchas cosas.

Este sistema está pensado para poder reutilizar de la forma más sencilla posible los componentes de cualquier aplicación (incluso de las aplicaciones básicas de Android, ya que los desarrolladores tienen acceso total a la API utilizada por las mismas), y de esta forma cualquier aplicación puede hacer públicas sus características para que otras aplicaciones puedan utilizarlas (sujetas a posibles restricciones de seguridad).

En este nivel, situado entre las aplicaciones de usuario y las librerías del sistema, encontramos los siguientes servicios disponibles:

- **Gestor de Actividades:** se encarga de gestionar el ciclo de vida de las aplicaciones así como de establecer un sistema de navegación entre las mismas.
- **Gestor de Ventanas:** servicio que se encarga de ofrecer una interfaz de acceso al gestor de ventanas del propio sistema operativo.
- **Proveedores de Contenido:** que permiten a una aplicación acceder a los datos de otras aplicaciones, como por ejemplo los datos de los contactos, y a su vez compartir los datos de la propia aplicación con el resto.
- **Sistema de Vistas:** Ofrece un gran número de vistas que pueden ser usadas para desarrollar las aplicaciones. Las vistas disponibles van desde las listas o las galerías pasando por formularios para introducir texto, botones o un navegador web.
- **Gestor de Paquetes:** permite obtener información relativa a las aplicaciones que están instaladas actualmente en el dispositivo.
- **Gestor de Telefonía:** provee de acceso a la información relativa a los servicios de telefonía del dispositivo como por ejemplo el estado de los mismos (si se está realizando una llamada o no, etc.).
- **Gestor de Recursos:** da acceso a los recursos usados por las aplicaciones como por ejemplo las imágenes, las cadenas de texto mostradas o los archivos XML en los que se especifica el diseño de la interfaz.
- **Gestor de Localización:** este componente permite por ejemplo que las aplicaciones obtengan información sobre la localización geográfica del dispositivo y puedan lanzar algún evento en función de esta información.
- **Gestor de Notificaciones:** permite por ejemplo que una aplicación muestre una notificación en la barra de estrado.

- **Servicio XMPP:** API que sirve para poder acceder a este servicio de intercambio de mensajes en formato XML.

3.3.3 Librerías

El conjunto de librerías de las que se compone Android es muy variado y extenso. Por un lado encontramos las librerías base de Android, escritas en Java y que ofrecen un conjunto de funcionalidades comunes a cualquier sistema operativo que pueden ser utilizadas en tiempo de ejecución por las aplicaciones. Estas librerías se encuentran empaquetadas en el archivo "android.jar" que se distribuye junto al SDK de Android.

Por otro lado, Android incluye un conjunto de librerías escritas en C o C++ que son usadas por los distintos componentes que lo integran y que además pueden ser utilizadas por las aplicaciones a través del framework de aplicaciones. A continuación se describen las principales características de algunas de las librerías del sistema:

- **Librerías del sistema "Bionic libc":** desarrolladas a partir de las librerías estándar de C creadas por el organismo BSD y modificadas especialmente para sistemas empotrados basados en Linux.
- **Librerías multimedia:** basadas en las librerías de OpenCore soportan la reproducción y grabación de un gran número de formatos de audio y video, así como de imágenes, entre los que se incluyen MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- **Librerías "Surface Manager":** estás librerías, usadas por el gestor de ventanas del nivel superior de la arquitectura de Android, gestionan el acceso a los servicios del sistema relacionados con la composición de ventanas para las aplicaciones que estén activas en algún momento.
- **WebKit:** es un motor de renderizado utilizado en los navegadores web que ofrece una serie de características básicas que permite el visionado de las páginas web y otras funcionalidades como por ejemplo mantener un historial.
- **SGL:** es la librería utilizada por Android para dibujar gráficos en dos dimensiones, texto o imágenes. Las siglas se corresponden con los términos en inglés Skia Graphics Library, siendo Skia la compañía encargada del desarrollo de esta librería escrita en C++ y distribuida bajo una licencia de código abierto.

- **Librerías 3D:** las cuales utilizan el API Open GL para sistemas empotrados y permiten que los dispositivos muestren gráficos tridimensionales haciendo uso de hardware específico para la aceleración de gráficos 3D (cuando el terminal lo posea) o bien mediante software.
- **FreeType:** estas librerías son utilizadas como motor de renderizado de las distintas fuentes disponibles en el sistema.
- **SQLite:** motor de base de datos disponible para todas las aplicaciones que se caracteriza por ser ligero y muy competo, implementando la mayor parte del estándar SQL-92.
- **Librerías SSL:** posibilitan la utilización de este protocolo para establecer comunicaciones seguras.

3.3.4 Maquina Virtual Dalvik

Dalvik es la máquina virtual utilizada por los dispositivos móviles basados en Android. La mayor diferencia de Dalvik frente a otras máquinas virtuales, o con la propia máquina virtual de Java, es que su arquitectura está basada en registros en contrapartida con la arquitectura basada en pila de estas otras máquinas virtuales. Cada aplicación de Android ejecuta su propio proceso, con una instancia propia de la máquina virtual, no obstante Dalvik ha sido diseñada para que un dispositivo pueda ejecutar varias instancias de la maquina virtual de una forma muy eficiente.

Entre otros cambios introducidos con el objetivo de optimizar la máquina virtual Dalvik, esta se ha diseñado para ocupar poco espacio en menoría, como consecuencia de estar pensada para ser utilizada por dispositivos que no disponen de una gran cantidad de memoria o velocidad de procesamiento.

Mientras que una JVM ejecuta los bytecode de cualquier clase compilada, en Dalvik las clases que componen una aplicación son convertidas, utilizando una herramienta incluida en el SDK llamada *dx*, desde el formato .class a un formato especial denominado Dalvik Executable (.dex) siguiendo el esquema que se puede ver en la siguiente figura:

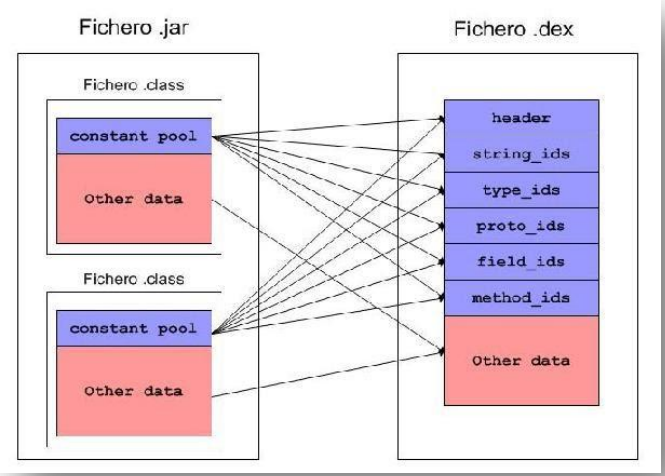

Figura 8. Correspondencia ficheros .class y .dex

La figura anterior nos muestra como en los ficheros ejecutables de Dalvik se agrupan los datos de las clases que componen la aplicación con el objetivo anteriormente mencionado de ocupar una cantidad menor de memoria, ya que no se almacenan los datos que puedan estar duplicados en las distintas clases. Por otra parte una vez instalados en un dispositivo, los ficheros ejecutables pueden sufrir otras modificaciones como por ejemplo que se altere el orden de los bytes de ciertos datos con el fin de optimizar aún más el rendimiento del terminal cuando ejecute la aplicación.

Otra característica de Dalvik es la inclusión de un compilador *Just in Time* desde la versión 2.2 de Android, pese a la negativa inicial por parte de Google para incluir esta técnica. Gracias a la compilación JIT, también conocida como traducción dinámica, el rendimiento de las aplicaciones en Android ha aumentado entre dos y cinco veces [6] respecto de la versión 2.1.

Por último, cabe mencionar que la máquina virtual Dalvik depende del Kernel para ciertas funcionalidades como la gestión de los hilos o de la memoria.

3.3.5 Kernel Linux

Como se mencionó anteriormente, el "cerebro" de Android se basa en el kernel de Linux 2.6.X (la versión suele ser actualizada con cada nueva iteración de Android) sirviendo como una capa de abstracción entre el hardware y el resto de capas de la arquitectura de Android.

A parte de esto, el kernel está encargado de gestionar los servicios de seguridad, la gestión de la memoria, de los procesos, del protocolo de red o los drivers del sistema, entre otras funciones.

3.4 Aplicaciones

Una de las características principales de Android es que cualquier aplicación puede utilizar componentes de otras, siempre y cuando cuente con los permisos adecuados. Por este motivo el sistema debe ser capaz de iniciar cualquier aplicación cuando sea preciso, y gracias a que en Android una aplicación no tiene un único punto de entrada, el sistema puede instanciar los componentes necesarios en cada momento.

Por defecto, cuando se inicia una aplicación, Android asigna a esta un proceso de Linux que cuenta con un único hilo de ejecución, por lo que todos sus componentes serán ejecutados en ese proceso e hilo. Sin embargo, se puede asignar un proceso diferente para cada componente así como lanzar nuevas hebras de ejecución para cada proceso.

En el siguiente apartado serán descritos los cuatro tipos de componentes de los que puede constar una aplicación en Android, así como el mecanismo de activación de los mismos y su ciclo de vida.

3.4.1 Componentes

Las aplicaciones de Android pueden estar compuestas por uno o más de los siguientes componentes descritos a continuación:

- **Actividades:**

Las actividades se caracterizan por presentar una interfaz visual con la que los usuarios pueden interactuar para realizar una tarea. Una aplicación puede consistir de una sola actividad o de varias como por ejemplo una aplicación de mensajería en la que se puede tener una actividad para enviar los mensajes, otra para listar los contactos y otra para cambiar los parámetros de configuración.

Cuando las aplicaciones cuentan con varias actividades, las cuales son totalmente independientes, se suele marcar una de ellas como la principal de forma que cuando se inicia la aplicación es esta actividad la que se presentan en primer lugar al usuario, y en el momento que se quiere cambiar a otra actividad sólo es necesario iniciarla desde la actual.

El sistema asigna una ventana para cada actividad aunque puede utilizar ventanas adicionales para mostrar por ejemplo una ventana emergente que requiere la interacción del usuario. El contenido visual de cada ventana está definido por una jerarquía de vistas, en la que cada una de ellas controla una zona de la interfaz y maneja las respuestas a las acciones del usuario dirigidas a esa zona en concreto.

- **Servicios:**

Los servicios se caracterizan principalmente por dos cosas: la primera, que no presentan una interfaz visual de usuario, y segundo, que la tarea para la que están programados es ejecutada en segundo plano. Por ejemplo un reproductor de música puede hacer uso de un servicio que reproduzca la música mientras el usuario atiende otras actividades.

Cuando iniciamos un servicio (o lo registramos para poder utilizarlo) se nos ofrece una interfaz de acceso al servicio que nos permite comunicarnos con el mismo, en el caso de un reproductor de música esta interfaz ofrecerá métodos para controlar la reproducción como la pausa o el rebobinado.

- **Receptores de difusión**

Los receptores de difusión son componentes con la función de recibir mensajes y reaccionar ante estos. Por ejemplo, cuando la batería del dispositivo está baja este componente informa al usuario.

Si bien no presentan una interfaz gráfica, si pueden invocar a una actividad en respuesta al mensaje recibido o incluso utilizar el gestor de notificaciones para alertar al usuario por ejemplo mediante una notificación en la barra de estado.

- **Proveedores de contenido**

La principal función de estos componentes es poner a disposición de otras aplicaciones los datos que estamos utilizando en nuestras aplicaciones. Estos datos suelen estar almacenados en bases de datos SQLite.

3.4.2 Iniciar y finalizar componentes

Android presenta varios mecanismos para iniciar y finalizar los distintos componentes de los que se compone una aplicación. Mientras que los proveedores de contenido son activados cuando se ejecuta una consulta sobre los datos que contienen, el resto de componentes descritos en el apartado anterior son activados a través de mensajes asíncronos llamados *Intents*, que son mandados al componente en cuestión mediante llamadas a funciones presentes en el API de Android.

Dependiendo del tipo de componente a activar, estos mensajes pueden contener información a cerca de una acción a llevar a cabo, además de especificar la URI de los datos sobre los que se va a aplicar dicha acción; o también pueden contener información acerca de un evento ocurrido en el terminal.

A la hora de finalizar los componentes de una actividad se debe tener en cuenta que los proveedores de contenido están activos mientras estén atendiendo una consulta y que por su parte los receptores de difusión permanecen activos mientras están respondiendo a un mensaje. Por lo tanto estos componentes no necesitan ser desactivados, mientras que las actividades y los servicios cuentan con mecanismos para finalizar su ejecución cuando así se requiera mediante funciones de la API de Android.

Así mismo, Android puede finalizar un componente si este no está siendo utilizado o es requerida más memoria para otros componentes.

3.4.3 Ciclo de vida de los componentes

En esta sección se comentarán los principales aspectos del ciclo de vida de los componentes de los que puede constar una aplicación.

- **Actividad**

En una actividad existen fundamentalmente tres estados: *activa* o *en ejecución* (cuando se encuentra en el primer plano de la pantalla, siendo la actividad el objetivo de las acciones del usuario), en *pausa* (si se ha perdido el foco de acción, pero sigue siendo visible para el usuario) y *detenida* (si la actividad ha sido ocultada completamente por otra). Si una actividad está en pausa o detenida, el sistema puede terminar su ejecución, ya sea porque le pide finalizar o simplemente porque elimina su proceso, teniéndose en cuenta ciertos aspectos que serán comentados más adelante en esta sección.

La transición de un estado a otro se registra con las llamadas a los métodos *onCreate*, *onStart*, *onRestart*, *onResume*, *onPause*, *onStop y onDestroy.* Todos estos métodos se utilizan para realizar tareas específicas cuando cambia el estado de la actividad, como el caso de *onCreate* el cual deben implementar todas las actividades y que se utiliza para aplicar la configuración inicial de una nueva instancia de la actividad, o el método *onPause* que puede ser utilizado para guardar los cambios realizados sobre los datos.

En conjunto, estos métodos definen el ciclo de vida completo de una actividad, en el cual podemos diferenciar tres bucles de ejecución:

- **Ciclo de vida completo** de una actividad que sucede entre la primera llamada a *onCreate* y la llamada final a *onDestroy*. La actividad lleva a cabo su configuración inicial en la llamada a *onCreate*, y libera todos los recursos que quedan en o*nDestroy*.
- **Tiempo de vida visible** de una actividad que ocurre entre una llamada a *onStart* hasta que se produce una llamada a *onStop*. Durante este tiempo, el usuario puede ver la actividad en pantalla, aunque puede que no sea en primer plano (interactuando con el usuario). Entre estos dos métodos, se pueden mantener los recursos que se necesitan para mostrar la actividad para el usuario.
- **Tiempo de vida en primer plano** de una actividad que ocurre entre una llamada a *onResume* hasta que se produce una llamada a *onPause*. Durante este tiempo, la actividad se encuentra en primer plano y el usuario está interactuando con la misma.

El siguiente diagrama ilustra estos bucles y los caminos que una actividad puede tomar entre los distintos estados. Los óvalos de color son los estados más importantes en los que se puede encontrar la actividad. Los rectángulos representan los métodos que se puede implementar para realizar operaciones cuando la actividad transita entre estados.

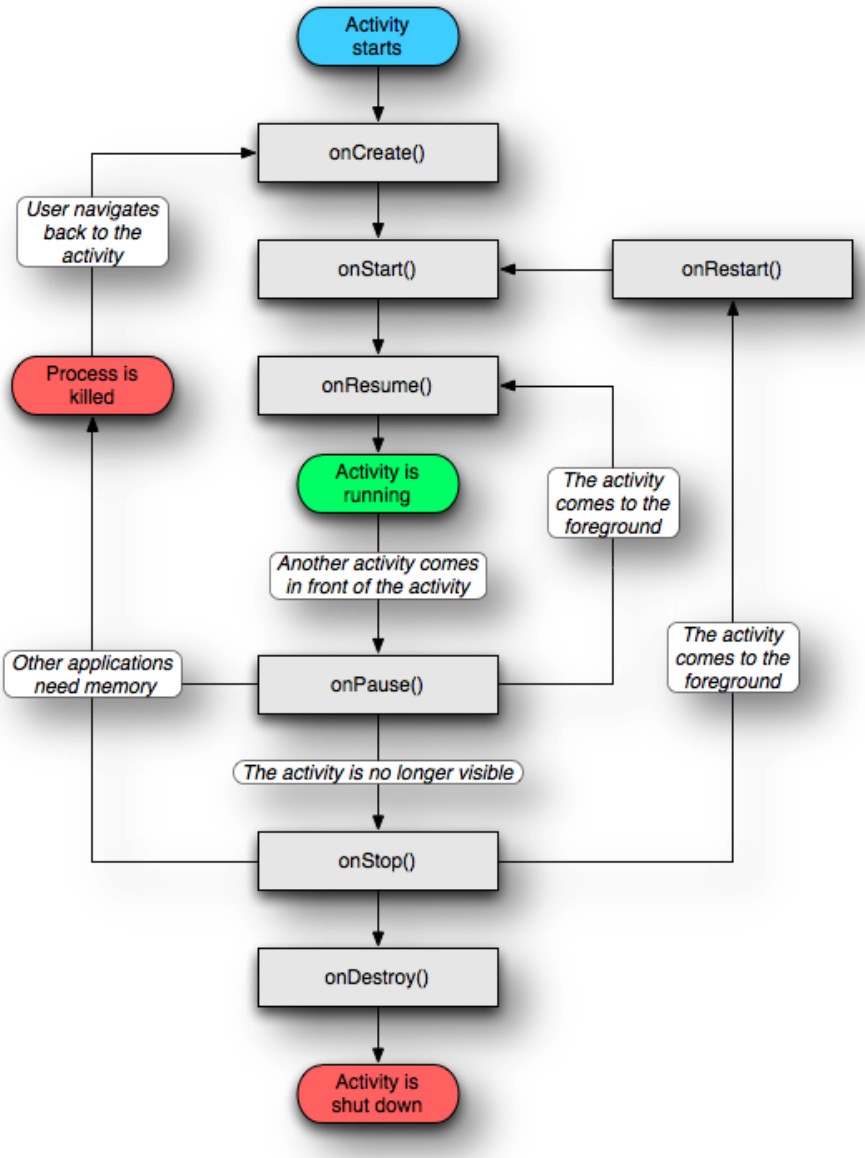

Figura 9 Ciclo de vida de una actividad

Como se mencionó en esta sección cuando el sistema finaliza una actividad se debe tener en cuenta que él cuando el usuario quiera volver a interactuar con dicha actividad esta debe presentarse tal y como se encontraba antes de ser expulsada de memoria por el sistema.

Para ello Android ofrece métodos dentro de su API que guardan el estado de la aplicación antes de que pueda ser eliminada (concretamente antes de la llamada a *onPause*) y recuperan el mismo cuando la actividad vuelve a tener el foco principal; aparte de objetos como *Bundle* con el que el estado de la aplicación es guardado como pares de nombre-valor.

- **Servicio**

Se puede acceder a un servicio de dos formas: iniciándolo y dejándolo ejecutar en segundo plano hasta que alguien lo pare o hasta que el mismo servicio finalice su ejecución con las funciones *startService* y *stopService*; y en segundo lugar, puede ser accedido programáticamente usando una interfaz definida y exportada por el servicio, utilizando los métodos *bindService* y *unbindService*.

Al igual que las actividades, los servicios tienen unos métodos que definen su ciclo de vida, con los que se puede monitorizar las transiciones de un estado a otro, que son: *onCreate*, *onStart*, y *onDestroy*. De esta manera se definen dos bucles de ejecución para estos componentes:

- **Ciclo de vida completo:** queda definido entre la primera llamada que se hace a *onCreate* y el instante en que finaliza la llamada a *onDestroy*.
- **Ciclo de vida activo:** comienza con la llamada a *onStart*.

Mientras que *onCreate* y *onDestroy* son llamadas comunes a los servicios independientemente de cómo sean iniciados, *onStart* solo puede ser invocado por los servicios que fueron iniciados por *startService*. Así mismo si un servicio permite a otros componentes que accedan a él existen otros métodos que definen el estado de un servicio, y son: *onBind*, *onUnbind* y *onRebind*.

El siguiente diagrama muestra las transiciones entre estados de un servicio. Aunque en el diagrama se separan los servicios que son creados con una llamada a *startService* de los que son creados con una llamada a *bindService*, se debe tener en cuenta que un servicio puede permitir a otros componentes asociarse con él, por lo que cualquier componente de este tipo puede recibir llamadas *onBind* y *onUnbind*.

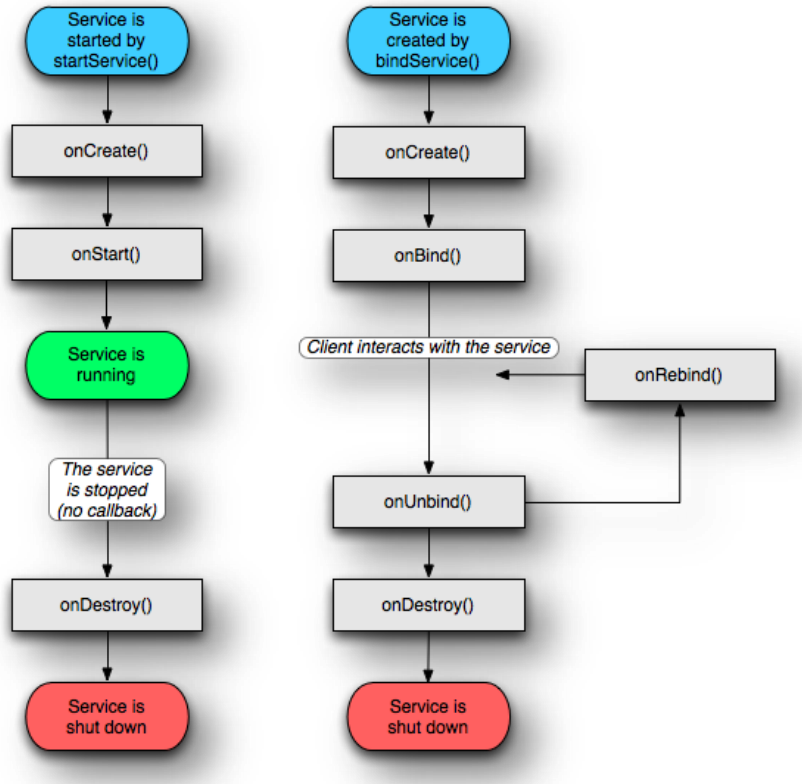

Figura 10. Bucle de ejecución de un servicio

- **Receptores de difusión**

Estos componentes tienen un solo método asociado con el ciclo de vida que es *onReceive*. Cuando llega un mensaje a un receptor de difusión se llama a este método pasándole dicho mensaje. En este momento, y sólo mientras ejecuta este método, el receptor se considera que está activo y por tanto un proceso que contenga a uno de estos componentes activos no podrá ser eliminado de la memoria del sistema.

- **Procesos**

A la hora de determinar qué proceso debe ser eliminado para ceder sus recursos a otro, Android sitúa cada proceso dentro de una jerarquía de importancia basada en el número de componentes que están ejecutándose y el estado de los mismos, siendo eliminados primero los procesos con menor prioridad en la jerarquía para después ir suprimiendo los procesos de mayor prioridad si es necesario.

Esta jerarquía consiste en cinco niveles ordenados a continuación de mayor a menor importancia:

- **Primer plano:** aquellos procesos que son requeridos por la actividad del usuario. Están en este nivel de la jerarquía los procesos que cumplen una de las siguientes condiciones:

  - Están ejecutando una actividad con la que el usuario esta interactuando
  - Contiene un servicio con el que el usuario esta interactuando
  - Contiene un servicio que está ejecutando alguna de sus llamadas de ciclo de vida (*onCreate*, *onStart*, *onDestroy*)
  - Contiene un receptor de difusión que está ejecutando el método *onReceive*.

En un mismo instante existen pocos procesos de este nivel de la jerarquía en memoria, los cuales son eliminados como último recurso cuando el sistema se ve obligado a realizar una re paginación de la memoria, por lo que la eliminación de estos procesos ayuda a que la interfaz responda a las acciones del usuario de forma apropiada.

- **Visibles:** son los procesos que no tienen componentes en primer plano pero que aún son visibles por el usuario. Cumplen alguna de estas condiciones:
  - Contiene una actividad que no está en primer plano pero aún es visible por el usuario (se ha invocado el método *onPause*)
  - Contiene un servicio que está enlazado a una actividad que es visible

Estos procesos no son eliminados a no ser que sea necesario para que los procesos de primer plano sigan ejecutando.

- **Servicios:** son los procesos que se encuentran ejecutando un componente de este tipo pero no se ajustan a las prioridades superiores. No son visibles para el usuario aunque suelen llevar a cabo tareas importantes para este (como reproducir música), por esto el sistema los mantiene en memoria hasta que es necesario que sean eliminados debido a que los procesos en primer plano o los visibles necesitan más memoria.
- **Segundo plano:** son los procesos que contienen actividades que no son visibles para el usuario (el método *onStop* ha sido invocado). Se pueden eliminar en cualquier momento de forma que los procesos de los niveles superiores de la jerarquía puedan obtener la memoria utilizada por los mismos. Normalmente hay varios procesos en segundo plano por lo que se mantiene una lista con los que contienen actividades que han sido usadas recientemente eliminándose por tanto en primer lugar los procesos más antiguos.
- **Vacios:** son los procesos que no contienen ningún componente activo y que únicamente se mantienen en memoria como cache para futuras

ejecuciones de forma que se mejore la rapidez con la que una aplicación inicia.

Android asigna siempre un proceso al nivel más elevado posible de prioridad, teniendo en cuenta ciertas consideraciones para ello, como por ejemplo la importancia de los componentes activos (en el caso de que un proceso contenga un servicio y una actividad visible, el proceso será asignado al nivel visible). Por otro lado la importancia de un proceso puede verse incrementada debido a que otros dependan del mismo, como por ejemplo en el caso de que un proceso este sirviendo a otro, el servidor nunca debe ser asignado a un nivel de importancia menor que el cliente.

Por último, en función de lo comentado a cerca del ciclo de vida de los procesos, es recomendable pensar que tipo de componente debe ejecutar cierta acción dentro de una aplicación. Por ejemplo, se debería asignar un servicio a una operación de descarga, la cual se supone va a consumir bastante tiempo, en lugar de una actividad para poder tener garantías de que la operación termina antes de que el proceso sea expulsado de la memoria.

3.5 Interfaz de usuario

En Android la interfaz de usuario se construye básicamente con dos componentes: *View* y *ViewGroup*.

Cada elemento *View* contenido en la interfaz de usuario almacena información acerca de la disposición y el contenido de una región específica de la pantalla, gestionando distintos aspectos como el tamaño, los cambios en el foco de atención o las interacciones con esa zona específica de la pantalla por parte del usuario. Los objetos *View* sirven de base a otros elementos llamados *widgets*, los cuales ofrecen funcionalidades ya implementadas y que pueden ser utilizadas por la interfaz de usuario de una actividad, como por ejemplo un campo de texto o un botón. Por su parte los objetos *ViewGroup* sirven para definir la disposición o *layout* de los elementos que componen la interfaz.

Para definir la interfaz de usuario se debe establecer una jerarquía de objetos *View* y *ViewGroup*, como por ejemplo la mostrada en la figura de la página siguiente, utilizando para ello un conjunto de *widgets* y *layouts*, que pueden estar predefinidos o creados desde cero por el desarrollador.

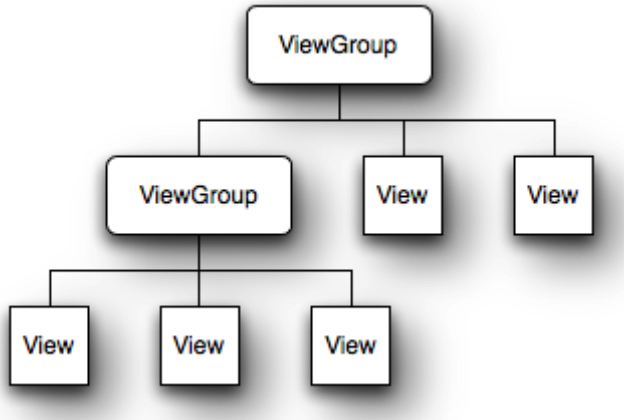

Figura 11. Jerarquía que define una interfaz de usuario

La forma más común de establecer esta jerarquía es con un archivo XML que contenga la disposición de los distintos elementos. Cada elemento especificado en este archivo es un objeto *View* o *ViewGroup* (o desciende de alguno de ellos), siendo los nodos hoja del árbol que representa esta jerarquía los objetos *View* y las ramas del mencionado árbol los objetos *ViewGroup*. Cada elemento tiene un nombre distintivo que representa una clase de Java que lo implementa. Por ejemplo un elemento <TextView> crea un objeto *TextView* en la interfaz, y un elemento <LinearLayout> crea un objeto ViewGroup equivalente. Por ejemplo, el siguiente fragmento de código crearía un ViewGroup en el que sus elementos, un cuadro de texto y un botón, se mostrarán uno a continuación del otro:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Figura 12. Archivo XML para definir layout

En el ejemplo anterior se utilizan dos *widgets* predefinidos de Android, *TextView* y *Button*, que como se comentó anteriormente son objetos de tipo *View* que sirven de interfaz para la interacción con el usuario. Las acciones llevadas a cabo son registradas por cada objeto *View* mediante una de las dos formas siguientes:

- Estableciendo "oyentes" dentro de cada objeto *View* que se dedique a manejar las acciones del usuario, como por ejemplo cuando se pulse un botón se definirá un método *OnClicklistener* que realizará una acción determinada.
- Redefiniendo los eventos desencadenados por la acción del usuario (cuando se implementa un nuevo objeto *View*).

Otro aspecto importante en una interfaz de usuario son los menús, ya que ofrecen una interfaz sencilla para realizar alguna acción dentro de la aplicación. La forma más común de mostrar un menú en Android es presionando el botón con el mismo nombre con el que cuentan todos los dispositivos diseñados para la plataforma, aunque también es habitual mostrar menús contextuales los cuales son mostrados cuando el usuario pulsa durante varios segundos un elemento de la interfaz.

Los menús al igual que los elementos que componen la interfaz de usuario se estructuran de forma jerárquica, pero normalmente son creados de forma distinta a dicha jerarquía, pues se definen en los métodos *onCreateOptionsMenu* y *onCreateContextMenu* dentro de las actividades, y en ellos se especifica los elementos que forman parte de cada menú, siendo el sistema el encargado de crear la estructura jerárquica oportuna y mostrar los elementos asignados al menú. Además, estos componentes de la interfaz de usuario manejan también sus propios eventos por lo que no es necesario definir "oyentes" que atiendan las acciones del usuario sobre los diferentes elementos del menú.

Por último, mencionar que Android permite una completa personalización de la interfaz de usuario pudiéndose aplicar estilos y temas visuales a los elementos de una aplicación, ya sean creados por el usuario o de los predefinidos por el sistema.

3.6 Recursos

Una buena práctica de programación en Android consiste en referenciar los recursos (imágenes, textos, etc.) de una aplicación de forma externa a la misma, de esta manera el mantenimiento de estos recursos es totalmente independiente a la aplicación. Otra ventaja de realizar esta práctica es que se pueden definir recursos alternativos para distintas configuraciones de dispositivo como el lenguaje, tamaño de pantalla, etc.

Para ello, por un lado es necesario organizar los recursos empleados por la aplicación en distintos directorios bajo el directorio "res" que es el directorio raíz asignado a estos elementos. De esta forma, las imágenes se almacenan en un directorio distinto al de las cadenas de texto o de los ficheros que contienen el *layout*

de una aplicación. Por otro lado, cuando se desea definir recursos alternativos, para por ejemplo una imagen, se debe crear un directorio que identifique la configuración alternativa con la que el recurso va a ser utilizado. Por ejemplo, en la siguiente imagen ala izquierda se muestra como un mismo recurso se utiliza independientemente de la configuración del dispositivo, mientras que en la figura ala derecha se ha definido un recursos para cuando la pantalla del dispositivo tiene una disposición de paisaje.

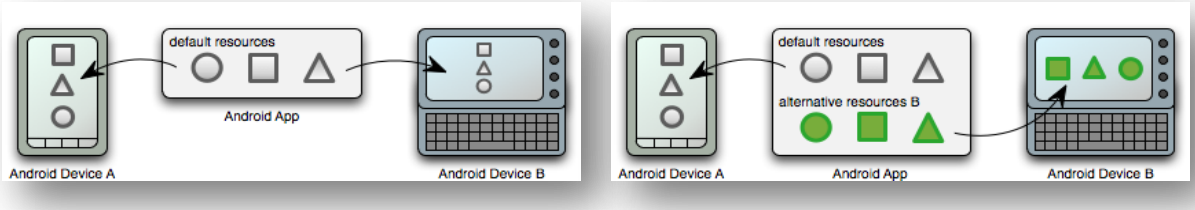

Figura 13. Utilización de recursos alternativos.

De esta forma el uso de recursos alternativos en una aplicación permite localizar una aplicación de forma sencilla definiendo varios archivos XML cada uno para un idioma.

3.7 Proveedores de contenido

Los proveedores de contenido almacenar y recuperan datos haciéndolos accesibles a todas las aplicaciones. Son la única manera de compartir datos entre aplicaciones, pues no hay un área de almacenamiento común al que todos los paquetes de Android pueden acceder.

Android incluye una serie de proveedores de contenido para los tipos de datos comunes como audio, vídeo, imágenes, información de contacto personal, etc.; siendo necesario para algunos de ellos tener el permiso adecuado para leer los datos que almacena, añadiendo dicho permiso en el archivo *AndoirdManifest* de la aplicación.

Por tanto, para compartir datos entre varias aplicaciones, se presentan dos opciones que involucran el uso de un proveedor de contenido: crear uno propio o agregar los datos a uno existente, siempre que el tipo de datos que se quiere almacenar sea compatible.

Para acceder a los datos que almacena un proveedor de contenido de forma independiente de como se almacenan estos, se define una interfaz de acceso común a cualquier proveedor a través de un objeto del API de Android llamado *ContentResolver* que ofrece una serie de métodos para interactuar con los datos almacenados. Cuando se realiza una consulta de los datos almacenados por un proveedor de contenido, el *ContentResolver* utiliza un objeto *Cursor* que permite navegar por la estructura tabular

en la que son almacenados los datos, similar a como se almacenan en una base de datos.

3.8 Seguridad y permisos

Android es un sistema multi-proceso, en el que cada aplicación (y partes del sistema) se ejecutan en su propio proceso. La mayoría de la seguridad entre las aplicaciones y el sistema se aplica en el nivel de proceso a través del estándar definidos por Linux, tales como la identificación de usuarios y grupos que se asignan a las aplicaciones. No obstante, se definen características adicionales de seguridad, de las que en los siguientes párrafos se detallaran ciertos aspectos, por un lado, a través de un mecanismo de "permisos" que impone restricciones a ciertas operaciones que un determinado proceso puede realizar, y por otro, concediendo permisos de acceso ad-hoc a ciertos datos mediante un mecanismo de URIs.

Un punto clave del diseño de la arquitectura de seguridad de Android es que, por defecto, ninguna aplicación tiene permisos para realizar operaciones que tengan un impacto negativo en otras aplicaciones, el sistema operativo, o el usuario. Esto incluye la lectura o escritura de datos privados del usuario (por ejemplo, contactos o mensajes de correo electrónico), leer o escribir archivos de otra aplicación, el acceso a la red, etc.

Por tanto para poder utilizar ciertas características de un dispositivo se deben incluir en el archivo *AndroidManifest.xml* una o varias etiquetas <uses-permission> en la que se identifica el permiso que la aplicación necesita. Por ejemplo en la siguiente imagen se muestra parte del archivo de manifiesto en el que se indican los permisos que la aplicación necesita (como acceso a internet o al dispositivo de almacenamiento externo) y que cuando se proceda a instalar la aplicación, será el instalador de paquetes quien otorgue los permisos solicitados por la misma.

```
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="7"/>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
</manifest>
```

Figura 14. Permisos necesarios para una aplicación

Al igual que se definen permisos de más alto nivel como los indicados en el párrafo anterior, también se pueden definir permisos específicos para un componente determinado o incluso definir permisos personalizados.

Por último hay veces en los que el sistema de permisos no es suficiente, por ejemplo cuando usamos un proveedor de contenido. Estos componentes quizás quieran protegerse de lecturas y/o escrituras sobre los datos que almacenan, como por ejemplo cuando una aplicación de correo electrónico que almacena los datos adjuntos recibidos, esos datos son accesibles mediante una URI, si esta dirección se le pasa a un visor de imágenes, este no debería ser capaz de acceder a los datos a no ser que se definen permisos por URI que dan acceso a estos datos.

3.9 El archivo AndroidManifest.xml

Este archivo XML con el que debe contar toda aplicación desarrollada para Android contiene información esencial acerca de la misma que el sistema debe conocer antes de poder ejecutar dicha aplicación. En este apartado se comentan alguna de las características de este archivo como pueden ser las funciones del mismo dentro de una aplicación o cómo se encuentra estructurado este archivo; recomendándose la lectura del apartado dedicado al mismo dentro de la guía de desarrollo de Android [13].

De este modo, el archivo de manifiesto es utilizado para entre otras cosas:

- Define el nombre del paquete APK que sirve como identificador único para la aplicación
- Describe los componentes de la aplicación –actividades, servicios, etc. Define además qué clases son las que implementan cada uno de estos componentes
- Define qué procesos albergarán los componentes de la aplicación
- Define los permisos que la aplicación debe poseer para acceder a determinados servicios o funciones del sistema o de otras aplicaciones
- Define los permisos que otras aplicaciones deben poseer para utilizar los componentes de la aplicación
- Define la versión mínima del API de Android que la aplicación necesita para funcionar
- Lista las librerías que la aplicación tiene enlazadas

La siguiente figura muestra la estructura general del archivo AndroidManifest.xml y los elementos que puede contener, los cuales son listados a continuación, no permitiéndose definir otros que no aparezcan en esta lista: <action>, <activity>, <activity-alias>, <application>, <category>, <data>, <grant-uri-permission>, <instrumentation>, <intent-filter>, <manifest>, <meta-data>, <permission>, <permission-group>, <permission-tree>, <provider>, <receiver>, <service>, <supports-screens>, <uses-configuration>, <uses-feature>, <uses-library>, <uses-permission> y <uses-sdk>.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Figura 15. Estructura general del archivo AndroidManifest.xml

Capitulo 4

Desarrollando para Android

4.1 Introducción

Una vez que se han estudiado los principales conceptos a cerca de la plataforma, es hora de ponerlos en práctica. En este capítulo se explicará cuales son las herramientas de desarrollo necesarias y como instalarlas y configurarlas. Además se mostrará cómo crear un proyecto desde cero, configurar el emulador, etc. Todo ello mientras se realiza un ejercicio sencillo que ponga en práctica todas las cuestiones mencionadas.

4.2 Configuración del entorno de trabajo

Para iniciar el desarrollo de una aplicación para Android, en primer lugar se debe configurar el entrono de trabajo. Para ello el equipo con el que se vaya a trabajar ha de contar con una serie de herramientas y librerías destinadas al desarrollo de aplicaciones para esta plataforma.

En primer lugar, es recomendable instalar el IDE Eclipse [12], aunque se puede trabajar con otros, el hecho de que el plug-in ADT este desarrollado principalmente para este IDE, hace pensar que la mejor opción para evitar posibles problemas de compatibilidad del plug-in, es utilizar este entorno de desarrollo.

Una vez instalado, el siguiente paso es obtener el SDK de Android [13] para la plataforma específica sobre la que se va a desarrollar la aplicación. Dado que el lenguaje de programación utilizado es Java, y este es independiente de la plataforma, se puede realizar el desarrollo en cualquiera de las plataformas domésticas habituales, es decir, Windows, Linux o Mac; si bien es necesario obtener la versión específica del SDK para la plataforma elegida.

A continuación deberemos instalar el plug-in para Eclipse ADT (Android Development Tools). Este plug-in permite crear y depurar aplicaciones de forma fácil y rápida añadiendo una serie de capacidades a Eclipse, que mejoran la experiencia de desarrollo, como las siguientes:

- Da acceso a otras herramientas de desarrollo de Android desde el interior del IDE Eclipse. Por ejemplo, ADT permite acceder a las distintas funciones de la herramienta MDDS como tomar capturas de pantalla, gestión de redirección de puertos, establecer puntos de interrupción, etc.
- Proporciona un asistente para nuevos proyectos, creando todos los archivos de base necesarios para una nueva aplicación Android
- Automatiza y simplifica el proceso de desarrollo de las aplicaciones
- Proporciona un editor de código que sirve de ayuda para escribir código XML válido para los archivos de recursos, interfaces de usuario o el archivo *manifest*
- Permite exportar el proyecto como un paquete APK y firmarlo digitalmente para ser distribuido a los usuarios

Para ello, una vez iniciado el mismo, dentro del menú ***Help*** se debe seleccionar la opción ***Install new software***.

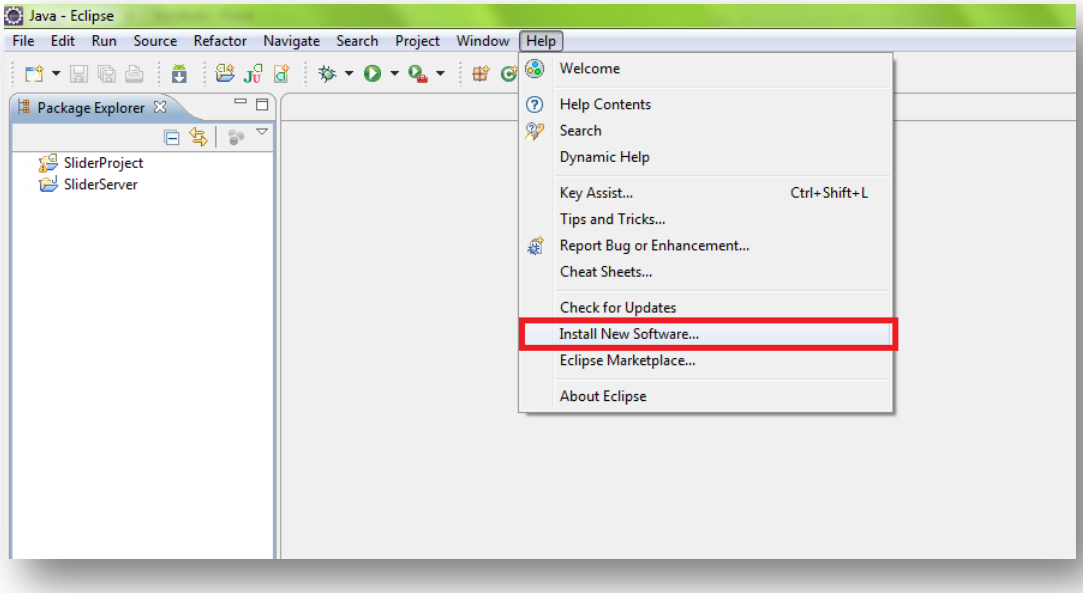

Figura 16. Configuración de Eclipse (I)

En el nuevo cuadro de dialogo que se abre se deberá pulsar el botón ***Add*** e indicar la URL desde donde vamos a obtener el plug-in [14].

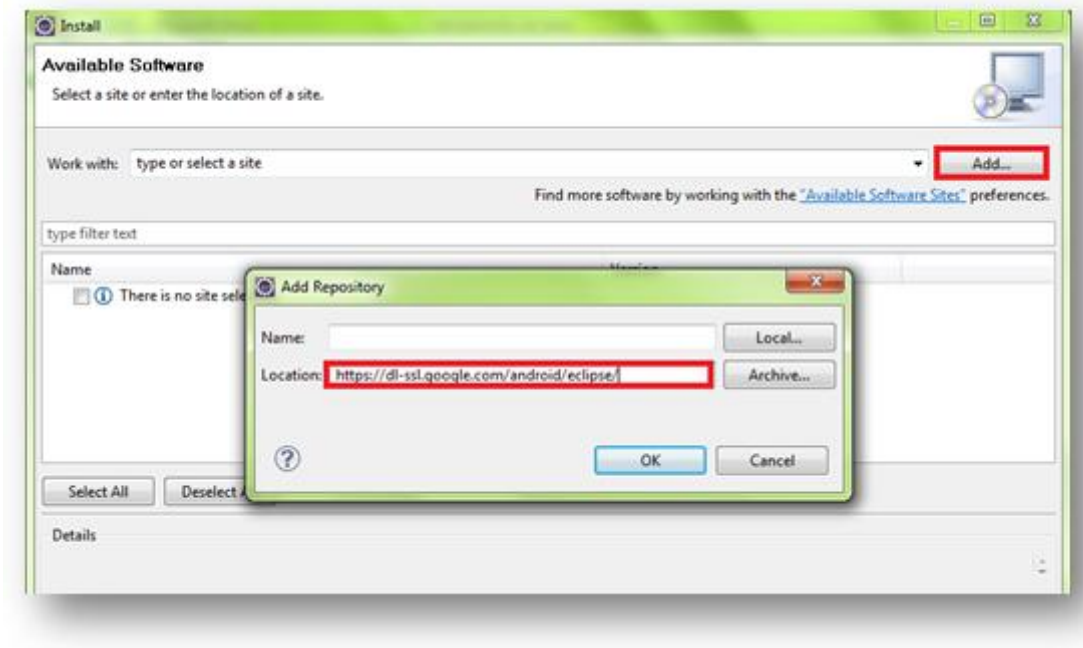

Figura 17. Configuración de Eclipse (II)

Una vez confirmada la URL del plug-in se podrá seleccionar que componentes van a ser instalados. En este caso se deben seleccionar los dos componentes que aparecen disponibles (Android DDMS y Android Development Tools).

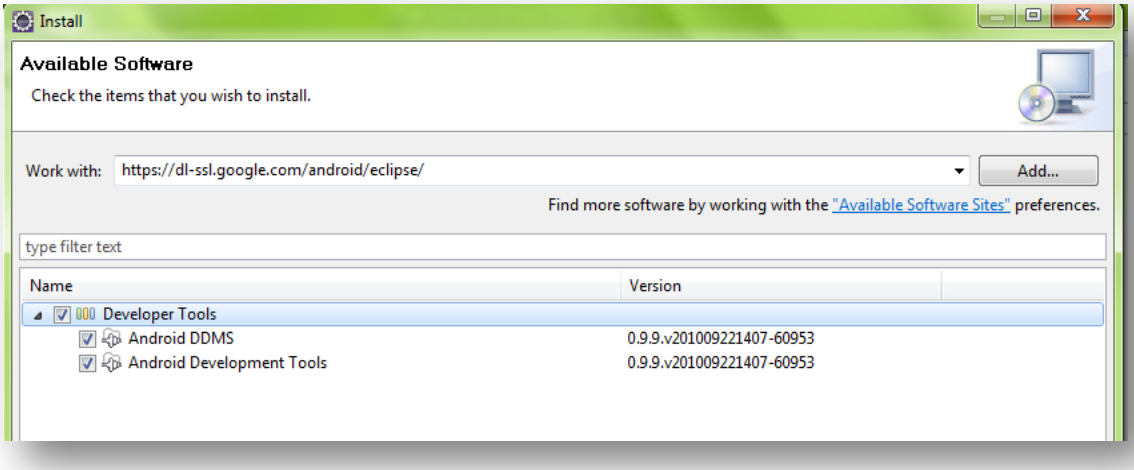

Figura 18. Configuración de Eclipse (III)

A continuación se deberá aceptar la licencia de uso y acto seguido comenzará la descarga e instalación de los componentes seleccionados. Por último, Eclipse debe ser reiniciado para poder comenzar a utilizar el nuevo plug-in instalado.

Tras instalar el plug-in y descargar el SDK de Android y descomprimido en cualquier ubicación del equipo de desarrollo, se le debe configurar el plug-in indicándole cual es el directorio en el que se encuentra el SDK. Para ello, desde la ventana principal del IDE se debe seleccionar en el menú ***Window*** la opción ***Preferences***.

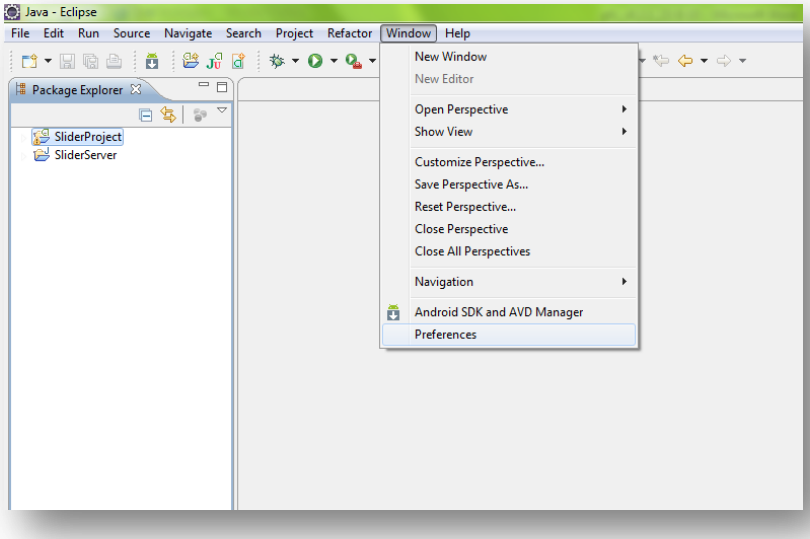

Figura 19. Configuración de Eclipse (IV)

En la ventana abierta, se debe seleccionar el apartado de Android en el que podremos indicarle la ubicación del SDK.

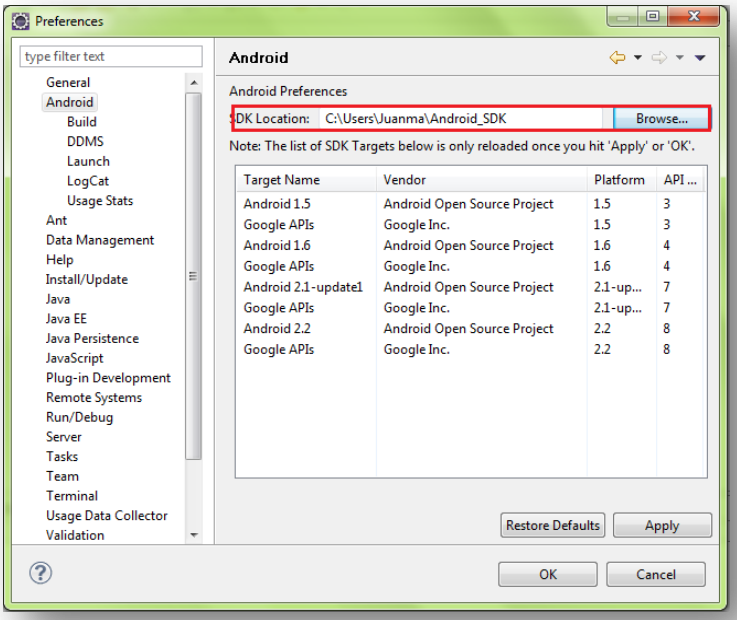

Figura 20. Configuración de Eclipse (V)

Una vez que el plug-in está configurado y listo para ser usado, la siguiente tarea a llevar a cabo será obtener los componentes de desarrollo que van a ser utilizados junto a ADT. Para ello, en el menú ***Window*** de Eclipse se debe elegir la opción ***Android SDK and AVD manager***, y en el nuevo cuadro de dialogo que aparece elegir ***Avaliable packages***.

En esta sección aparecerá un listado de los componentes disponibles para instalar.

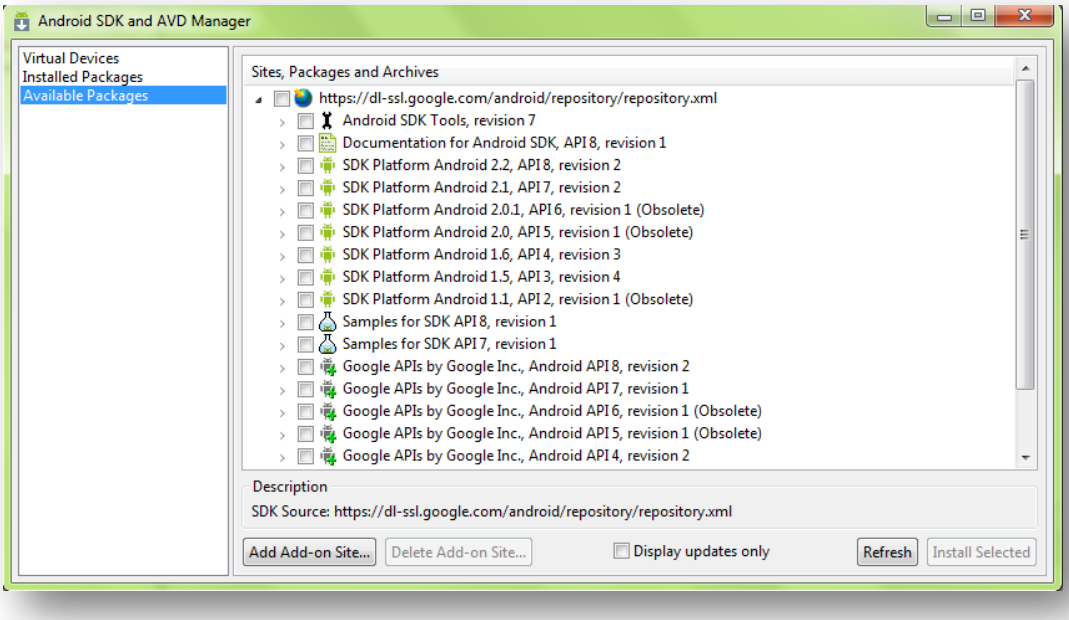

Figura 21. Configuración Eclipse (VI)

Para comenzar a desarrollar una aplicación es requisito indispensable contar con al menos una plataforma de desarrollo de nuestra elección. Para decidir entre una plataforma y otra es recomendable leer la información disponible en la página de desarrollo oficial de Android al respecto [13].

Opcionalmente se pueden obtener más componentes, de hecho es recomendable instalar también la documentación, el código de ejemplo y el driver USB para poder utilizar un dispositivo externo para realizar pruebas. Además, si la aplicación que se pretende desarrollar va a hacer uso de las librerías de mapas se debería instalar el API de Google que da acceso a las mismas.

Por último, es recomendable configurar las variables de entorno del sistema para incluir dentro de la variable PATH la ubicación de la carpeta *tolos* del SDK de Android, de forma que acceder a las herramientas ubicadas en ella sea más sencillo.

4.3 Creación de un proyecto

Una vez se ha configurado el entrono de trabajo adecuadamente, el siguiente paso del desarrollo será crear un proyecto. El plug-in ADT facilita esta tarea enormemente, pues incluye una herramienta para la creación de los mismos, con la que en unos pocos pasos se tendrá lista la estructura de carpetas y archivos necesarios para comenzar a trabajar.

Por tanto, para crear un proyecto, desde el menú de ***File*** se debe elegir la opción ***New*** y después ***Project***. En el nuevo cuadro de dialogo que aparece se debe seleccionar en la rama ***Android*** la opción ***Android Project*** y pulsar ***Next***.

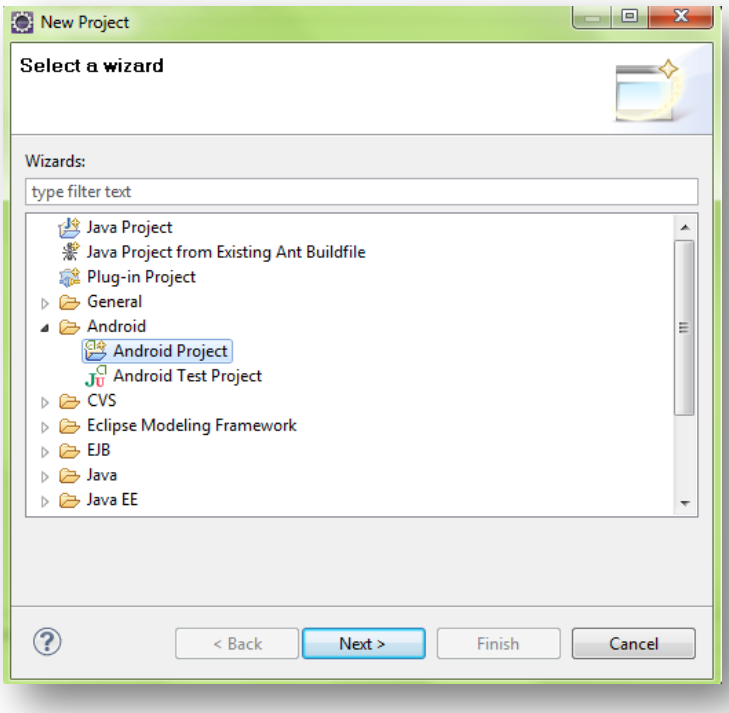

Figura 22. Creación de un proyecto (I)

A continuación se deberán indicar una serie de parámetros necesarios para la creación del proyecto. En la siguiente tabla se indica cuales de estos parámetros deberían suministrarse para crear un proyecto de forma satisfactoria:

ELEMENTO	DESCRPCIÓN
Project Name	El nombre del proyecto que vamos a crear.
Build Target	Indica la versión del API de Android que vamos a utilizar en el proyecto. Es recomendable utilizar una plataforma de desarrollo con la versión más baja posible, a no ser que se requiera una característica disponible únicamente en una versión más

	reciente del API de Android.
Application Name	Nombre de la aplicación que será mostrado en el dispositivo.
Package Name	Indica el espacio de nombres para los paquetes del código fuente de la aplicación (siguiendo las mismas convenciones que en el lenguaje de programación Java).
Create Activity	Si se selecciona esta opción, se creará una actividad con este nombre y será la actividad principal de la aplicación.
Min SDK Version	Indica la versión mínima de Android que debe tener instalado el terminal para poder ejecutar la aplicación.

Tabla 4. Parámetros de creación de un proyecto

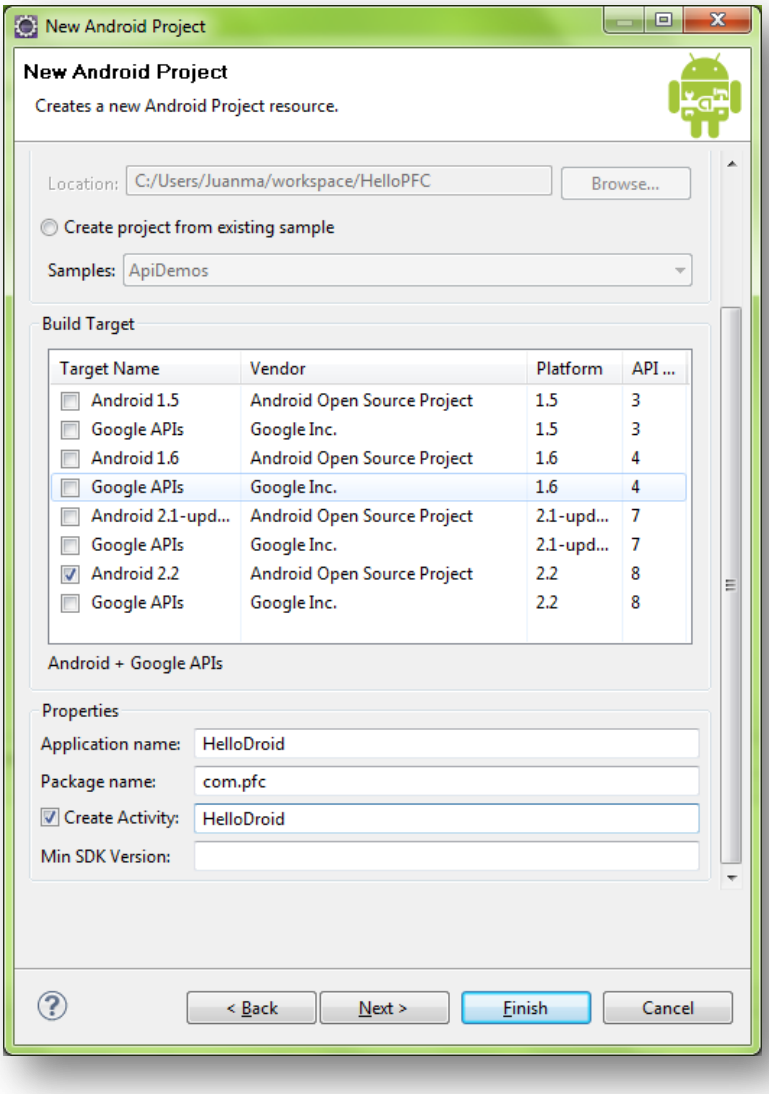

Figura 23. Creación de un proyecto (II)

Al finalizar el proceso de creación del proyecto, tal y como se puede observar en la siguiente figura, se han creado una serie de carpetas y archivos dentro del proyecto.

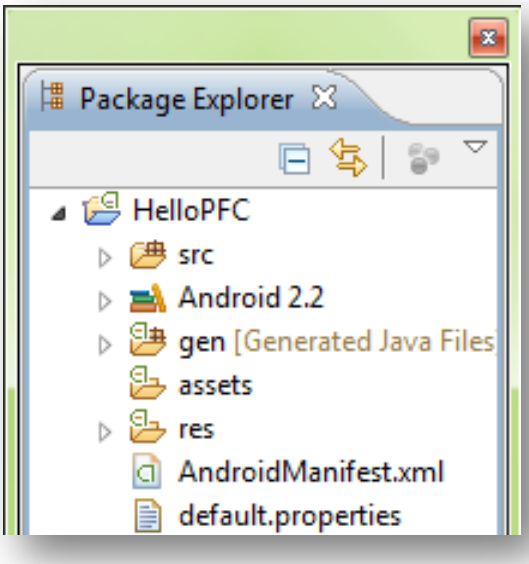

Figura 24. Creación de un proyecto (III)

En la siguiente tabla se encuentra explicado el uso de cada uno de estos elementos creados:

ELEMENTO	DESCRPCIÓN
src	Código fuente de la aplicación.
<Versión de Android> (por ejemplo Android 2.2/)	Archivo JAR que contiene las librerías de Android y cuya versión se especifica en el momento de creación del proyecto.
gen	Contiene los archivos Java creados automáticamente por el plug-in ADT como por ejemplo el archivo R.java.
res	Directorio en el que se almacenan los recursos de una aplicación como imágenes, texto, etc.
AndroidManifest.xml	El archivo de manifiesto del proyecto.
default.properties	Contiene los parámetros de configuración del proyecto.

Tabla 5. Estructura de un proyecto en Android

4.4 Desarrollo de una aplicación sencilla

Una vez que se ha creado el proyecto, lo siguiente será tomar contacto con Android y desarrollar una aplicación sencilla como puede ser el típico "¡Hola Mundo!". Más adelante en este documento se discutirán cuestiones más avanzadas sobre el desarrollo en Android cuando se explique el desarrollo de la aplicación desarrollada en este proyecto.

Por tanto como toma de contacto con la plataforma se ha decido explicar cómo crear una sencilla calculadora paso a paso.

En primer lugar se va a definir la interfaz de usuario de la actividad principal y única de esta aplicación. Para ello ADT ofrece una utilidad gráfica que puede ser bastante útil para definir el layout y añadir vistas simplemente arrastrando los elementos desde la lista situada a la izquierda al rectángulo que simula la pantalla del terminal, como se puede ver en la siguiente imagen.

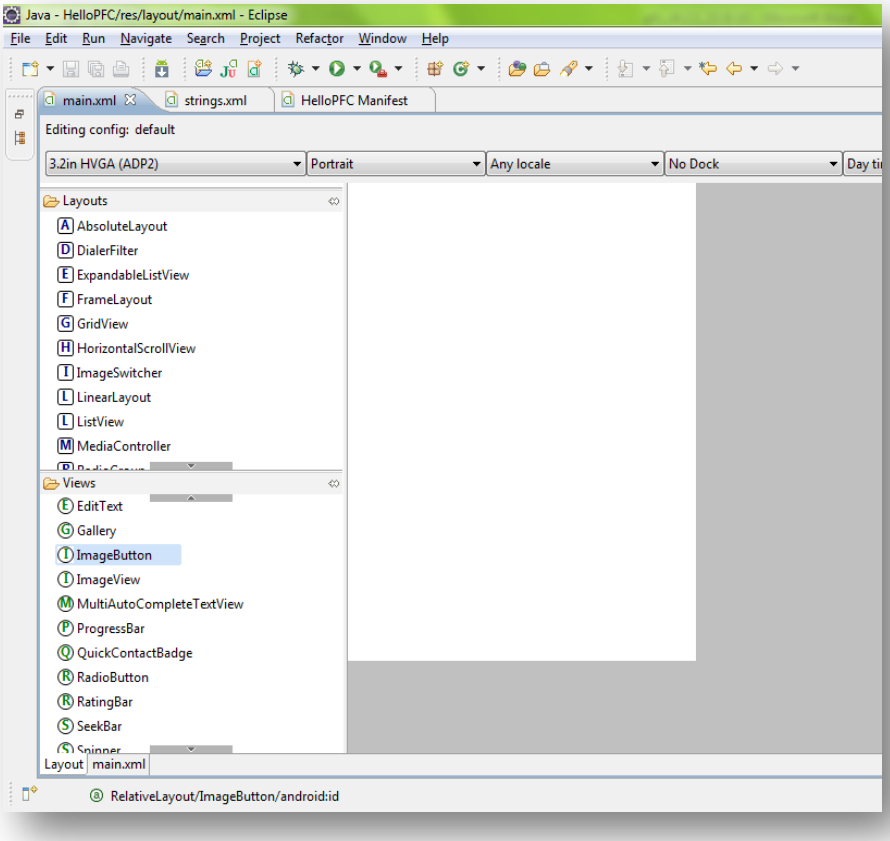

Figura 25. Creación interfaz de usuario

Otras veces resultará incluso más cómodo editar el archivo XML correspondiente para especificar los distintos atributos de cada elemento. Además en esta ventana podemos especificar otros atributos de la actividad en cuestión como el

tamaño en pulgadas de la pantalla, la orientación (apaisada o paisaje), el tema que se aplicará, la resolución, etc.

Para esta aplicación de ejemplo se ha definido una interfaz de usuario tal y como se puede ver en la siguiente figura, y de la que se van a comentar ciertos aspectos revisando el archivo XML asociado:

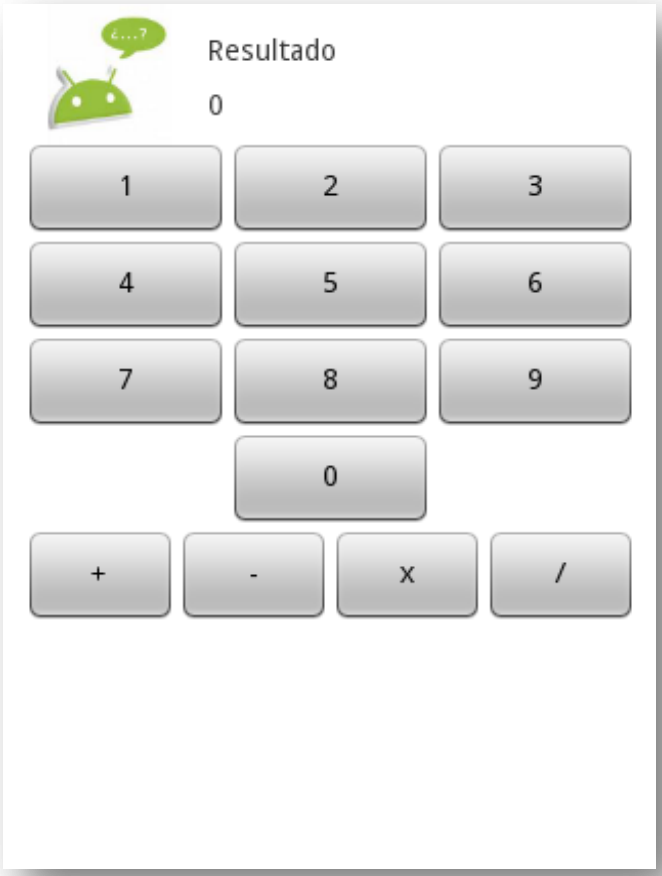

Figura 26. Interfaz de usuario de la aplicación

El archivo XML asociado a la interfaz mostrada en la figura anterior se corresponde con el código a continuación

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageButton android:id="@+id/ImageButton01"
        android:background="#FFFFFF" android:src="@drawable/pfc"
        android:layout_marginLeft="10px" android:layout_width="wrap_content"
        android:layout_height="wrap_content"></ImageButton>

    <TextView android:id="@+id/TextView00" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/resultado"
        android:layout_marginLeft="10px" android:layout_marginTop="15px"
        android:layout_toRightOf="@id/ImageButton01" />

    <TextView android:id="@+id/TextView01" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:layout_below="@id/TextView00"
        android:layout_marginLeft="10px" android:layout_marginTop="10px"
        android:layout_marginBottom="10px" android:layout_toRightOf="@id/ImageButton01" />

    <Button android:text="1" android:id="@+id/Button01"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/TextView01" android:layout_marginLeft="10px"></Button>
    <Button android:text="2" android:id="@+id/Button02"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/TextView01" android:layout_toRightOf="@id/Button01"></Button>
    <Button android:text="3" android:id="@+id/Button03"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/TextView01" android:layout_toRightOf="@id/Button02"></Button>

    <Button android:text="4" android:id="@+id/Button04"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button01" android:layout_marginLeft="10px"></Button>
    <Button android:text="5" android:id="@+id/Button05"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button01" android:layout_toRightOf="@id/Button04"></Button>
    <Button android:text="6" android:id="@+id/Button06"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button01" android:layout_toRightOf="@id/Button05"></Button>

    <Button android:text="7" android:id="@+id/Button07"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button04" android:layout_marginLeft="10px"></Button>
    <Button android:text="8" android:id="@+id/Button08"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button04" android:layout_toRightOf="@id/Button07"></Button>
    <Button android:text="9" android:id="@+id/Button09"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button04" android:layout_toRightOf="@id/Button08"></Button>

    <Button android:text="0" android:id="@+id/Button10"
        android:layout_width="100px" android:layout_height="wrap_content"
        android:layout_below="@id/Button08" android:layout_toRightOf="@id/Button07"></Button>

    <Button android:text="+" android:id="@+id/Button11"
        android:layout_width="75px" android:layout_height="wrap_content"
        android:layout_below="@id/Button10" android:layout_marginLeft="10px"></Button>
    <Button android:text="-" android:id="@+id/Button12"
        android:layout_width="75px" android:layout_height="wrap_content"
        android:layout_below="@id/Button10" android:layout_toRightOf="@id/Button11"></Button>
    <Button android:text="x" android:id="@+id/Button13"
        android:layout_width="75px" android:layout_height="wrap_content"
        android:layout_below="@id/Button10" android:layout_toRightOf="@id/Button12"></Button>
    <Button android:text="/" android:id="@+id/Button14"
        android:layout_width="75px" android:layout_height="wrap_content"
        android:layout_below="@id/Button10" android:layout_toRightOf="@id/Button13"></Button>

</RelativeLayout>
```

Figura 27. Archivo que define el layout de la aplicación

En el código mostrado en la figura anterior se han macado una serie de zonas, de las que se van a comentar ciertos detalles a continuación:

1. El *layout* de la aplicación va ser de tipo *RelativeLayout*, esto implica que los objetos View que contenga deberán ser ordenados explícitamente. Además se especifica que la orientación será vertical y con los atributos *layout_height* y *layout_width* el tamaño de la vista, que este caso particular rellenara toda la pantalla del dispositivo.

2. A continuación se pueden añadir los elementos de la interfaz con los que el usuario puede interactuar. Por ejemplo se puede añadir un *ImageButton* que tendrá la función de calcular el resultado de las operaciones. Como es un botón que muestra una imagen con la etiqueta *android:src* se le indica que imagen que debe mostrar. Con el resto de etiquetas se le indicará el tamaño, que en este caso se ajusta al tamaño de la imagen utilizando la constante *wrap_content*; y se establece un margen a la izquierda de diez pixeles.
3. El siguiente elemento es un campo de texto al que se le asigna una cadena de texto utilizando los recursos definidos en el archivo *strings.xml*. Además se especifica que debe estar colocado a la derecha de otro elemento, en este caso del *ImageButton* definido anteriormente.
4. Este otro campo de texto se utiliza para mostrar los resultados calculados. No se inicializa con ninguna cadena de texto. Se indica que debe situarse a la derecha del *ImageButton* y por debajo del campo de texto anterior con las etiquetas que aparecen en la figura.
5. El siguiente paso es definir los botones numéricos de la calculadora. Para ello se añaden al *layout* varios de estos elementos definiendo con la etiqueta *android:text* el texto mostrado por el botón. A cada botón se le indica mediante las etiquetas *layout_below* y *layout_toRightOf* en qué lugar de la interfaz debe aparecer y también se especifica un ancho para cada botón de 100 pixeles.
6. Siguiente fila de números, en este caso cuatro, cinco y seis.
7. Se añaden los números siete, ocho y nueve.
8. Por último se añade el cero. Como se ha indicado anteriormente, todos estos elementos se colocan en función a otros ya existentes en la interfaz mediante las etiquetas oportunas.
9. Por último se añaden los botones correspondientes a los operadores aritméticos para la suma, resta, multiplicación y división.

Se puede comprobar que el proceso de creación de una interfaz puede resultar bastante simple, dependiendo claro está de las necesidades de cada aplicación, pues tan solo se necesita elegir un *layout para* la misma y a continuación ir añadiendo *widgets* modificando sus atributos según convenga.

Una vez que se ha completado la interfaz, el siguiente paso es añadir funcionalidades a los elementos que la conforman. Por lo tanto se deberán editar los archivos del código fuente de la aplicación. Las siguientes figuras muestran los contenidos del fichero Java que ha sido editado para añadir funcionalidades a la interfaz.

```
package com.pfc;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;

public class HelloDroid extends Activity {

    private double resultado;
    private TextView resultado_view;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        resultado = 0;
        resultado_view = (TextView) findViewById(R.id.TextView01);

        // Acciones para los números
        Button b1 = (Button) findViewById(R.id.Button01);
        b1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("1");
            }
        });
        Button b2 = (Button) findViewById(R.id.Button02);
        b2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("2");
            }
        });
        Button b3 = (Button) findViewById(R.id.Button03);
        b3.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("3");
            }
        });
        Button b4 = (Button) findViewById(R.id.Button04);
        b4.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("4");
            }
        });
        Button b5 = (Button) findViewById(R.id.Button05);
        b5.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("5");
            }
        });
        Button b6 = (Button) findViewById(R.id.Button06);
        b6.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("6");
            }
        });
        Button b7 = (Button) findViewById(R.id.Button07);
        b7.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("7");
            }
        });
        Button b8 = (Button) findViewById(R.id.Button08);
        b8.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("8");
            }
        });
        Button b9 = (Button) findViewById(R.id.Button09);
        b9.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("9");
            }
        });
        Button b0 = (Button) findViewById(R.id.Button10);
        b0.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("0");
            }
        });
```

Figura 28. Código Java de la aplicación (I)

```
        // Acciones operadores
        Button bplus = (Button) findViewById(R.id.Button11);
        bplus.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("+");
            }
        });
        Button bminus = (Button) findViewById(R.id.Button12);
        bminus.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("-");
            }
        });
        Button bmul = (Button) findViewById(R.id.Button13);
        bmul.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("*");
            }
        });
        Button bdiv = (Button) findViewById(R.id.Button14);
        bdiv.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                resultado_view.append("/");
            }
        });

        ImageButton imb1 = (ImageButton) findViewById(R.id.ImageButton01);
        imb1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                CharSequence cq = resultado_view.getText();
                String res = new String(cq.toString());
                if (res.indexOf('+') != -1) {
                    resultado = new Double(res.substring(0, res.indexOf('+')))
                            + new Double(res.substring(res.indexOf('+') + 1));
                } else if (res.indexOf('-') != -1) {
                    resultado = new Double(res.substring(0, res.indexOf('-')))
                            - new Double(res.substring(res.indexOf('-') + 1));
                } else if (res.indexOf('*') != -1) {
                    resultado = new Double(res.substring(0, res.indexOf('*')))
                            * new Double(res.substring(res.indexOf('*') + 1));
                } else if (res.indexOf('/') != -1) {
                    resultado = new Double(res.substring(0, res.indexOf('/')))
                            / new Double(res.substring(res.indexOf('/') + 1));
                }
                resultado_view.setText(new Double(resultado).toString());
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        boolean result = super.onCreateOptionsMenu(menu);
        menu.add(0, 0, 0, "Limpiar").setIcon(android.R.drawable.ic_delete);
        menu.add(0, 1, 0, "Salir").setIcon(
                android.R.drawable.ic_menu_close_clear_cancel);
        return result;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case 0:
            resultado_view.setText("");
            resultado = 0;
            return true;
        case 1:
            this.finish();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Figura 29. Código Java de la aplicación (II)

En estas figuras podemos apreciar que se han definido tres métodos:

- El primero de ellos, onCreate, se utiliza como punto de entrada a la aplicación y en él se configuran todos los elementos de la interfaz. Primero se especifica el layout que debe utilizar la actividad, llamando al método *setContentView* para ello y pasándole la constante que identifica el layout

que ha sido generada de forma automática y almacenada en el fichero R.java. En segundo lugar se define el comportamiento de todos los botones de la calculadora que se está programando. Para realizar esto, es necesario "encontrar" el botón dentro de la interfaz utilizando el método *findViewById* al que se le debe indicar la id del elemento, y que se encuentra en el fichero R antes mencionado. Una vez que hemos obtenido el elemento se le asigna un evento que "escuchará" cuando se pulsa el botón realizando en ese momento alguna acción como puede ser mostrar el número pulsado o calcular una operación aritmética y devolver el resultado de la misma.

- El segundo método se utiliza para definir un menú con el que el usuario podrá realizar nuevas acciones. En este caso se añaden dos elementos al menú, el primero para limpiar el cuadro de texto donde se muestran los resultados y el segundo para salir de la aplicación.
- El tercer método mostrado es el que realmente lleva a cabo las acciones indicadas en el menú.

4.5 Creación de un dispositivo virtual

El siguiente paso en el desarrollo será ejecutar la aplicación que acabamos de desarrollar. Pero para poder hacer esto es necesario contar bien con un dispositivo de desarrollo preparado para esta tarea o si no ejecutar la aplicación en el emulador de Android creando un dispositivo virtual AVD (Android Virtual Device).

Para crear un AVD es necesario seguir estas indicaciones:

1. En Eclipse desde el menú ***Window*** seleccionar ***Android SDK and AVD Manager***, o seleccionar el botón que aparece en la barra de herramientas con el mismo nombre.
2. En la ventana que se abre, seleccionar el panel de ***Virtual Devices*** en la que aparecen los dispositivos que ya existen. Al pulsar sobre ***New*** se abrirá una nueva ventana en la que podemos indicar los parámetros del nuevo dispositivo virtual.

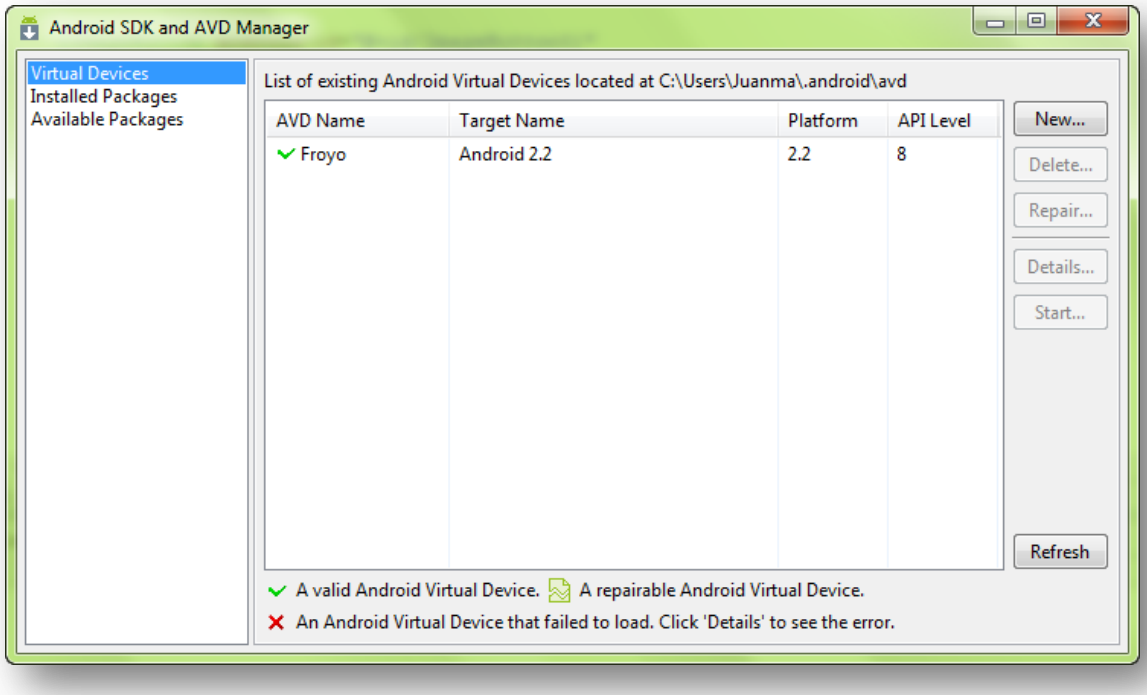

Figura 30. Creación AVD (I)

3. Los parámetros que pueden ser configurados para un dispositivo virtual se encuentran detallados en la siguiente tabla:

ELEMENTO	DESCRPCIÓN
Soporte para tarjeta SD	AVD soporta la emulación de una tarjeta SD.
Soporte para Dpad	AVD soporta la emulación de un pad digital.
Acelerómetro	AVD soporta la emulación de un dispositivo acelerómetro.
Máximo número de pixels de la cámara en horizontal	Resolución horizontal de la cámara emulada por el AVD.
Tamaño de la partición cache	Tamaño en MB de la partición cache.
Soporte de reproducción de audio	AVD soporta la reproducción de audio.
Soporte para track-ball	AVD soporta la emulación de un dispositivo track-ball.
Máximo número de pixels de la cámara en vertical	Resolución vertical de la cámara emulada por el AVD

Soporte para cámara	AVD soporta la emulación de un dispositivo de imagen.
Soporte para batería	AVD soporta la emulación del comportamiento de la batería.
Soporte para pantalla táctil	AVD soporta la emulación de una pantalla táctil.
Soporte de grabación de audio	AVD soporta la grabación de audio.
Soporte de GPS	AVD soporta la emulación de un dispositivo GPS.
Soporte para partición de cache	AVD soporta la emulación de una partición de cache en el sistema.
Soporte para teclado	AVD soporta la emulación de un teclado físico en el sistema.
Tamaño máximo ocupado en pila por una aplicación	Indica el tamaño máximo que una aplicación puede ocupar en la pila de la máquina virtual Dalvik.
Tamaño de la memoria RAM	Tamaño de la memoria RAM con la que cuenta el AVD.
Soporte para modem GSM	AVD soporta la emulación de un modem GSM.

Tabla 6. Parámetros configurables de un AVD

4. En la ventana que aparece en la siguiente figura se deben indicar de forma obligatoria las opciones ***Name*** y ***Target***, teniendo en cuenta que la opción *target* debe ser igual a la versión del API necesaria para ejecutar la aplicación.
5. Opcionalmente, se puede indicar un tamaño en megabytes de la tarjeta SD que va a emular el AVD o elegir un archivo previamente creado para este fin, así como especificar la resolución de pantalla y la densidad (píxeles por pulgada) de la misma.

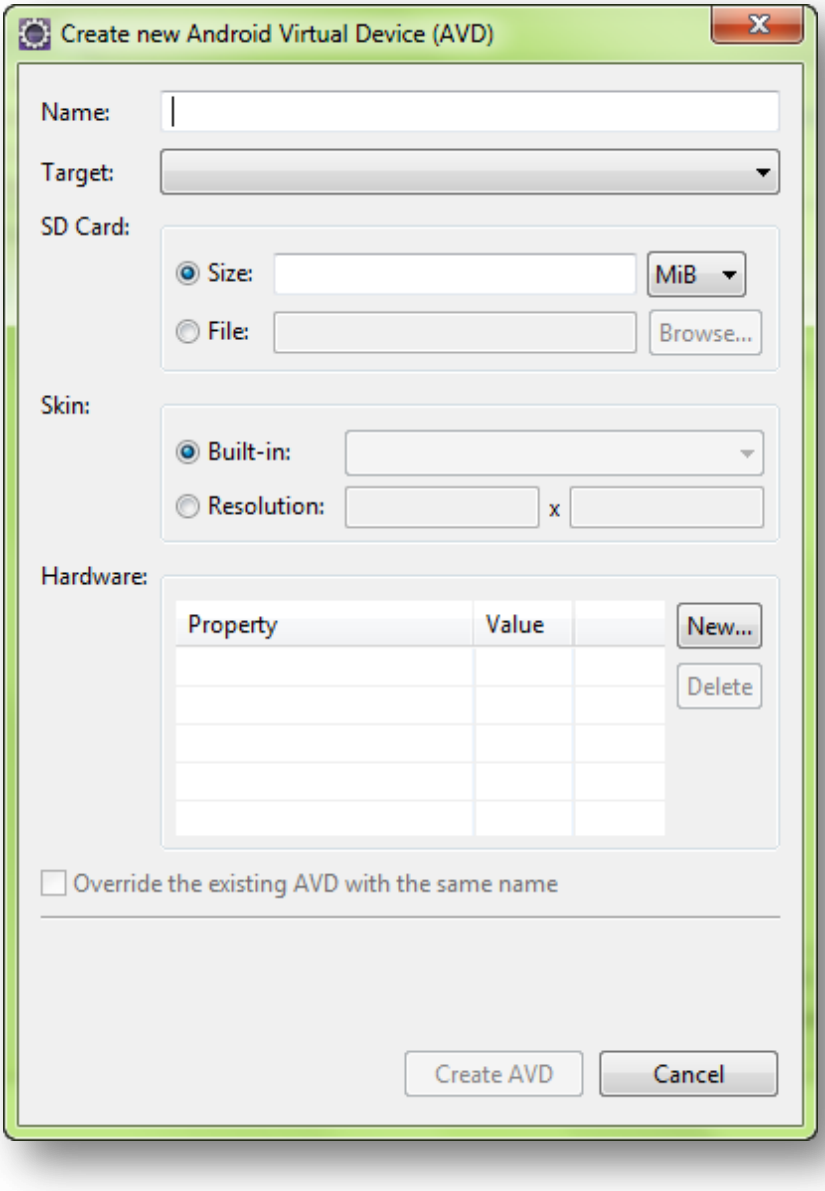

Figura 31. Creación AVD (II)

6. Tras configurar todos estos parámetros solo queda pulsar en el botón ***Create AVD***.

4.6 Ejecutar una aplicación

Como se comentó en el apartado anterior, una de las opciones para ejecutar una aplicación de Android es crear un AVD y lanzar el emulador con la configuración del dispositivo virtual creado.

Una vez que se ha creado este dispositivo virtual, para ejecutar la aplicación basta con pulsar el botón ***Run*** (o ***Debug***) de la barra de herramientas de Eclipse. Automáticamente Eclipse realizará las siguientes acciones:

1. Compilar el proyecto si existen cambios desde la última ejecución
2. Crear una configuración de ejecución por defecto para el proyecto, en el caso de que no se haya especificado una
3. Instalará e iniciará la aplicación en el emulador en base a la configuración indicada

Como resultado, aparecerá una nueva ventana en la que se está ejecutando la aplicación en el emulador, tal y como puede verse en la siguiente figura:

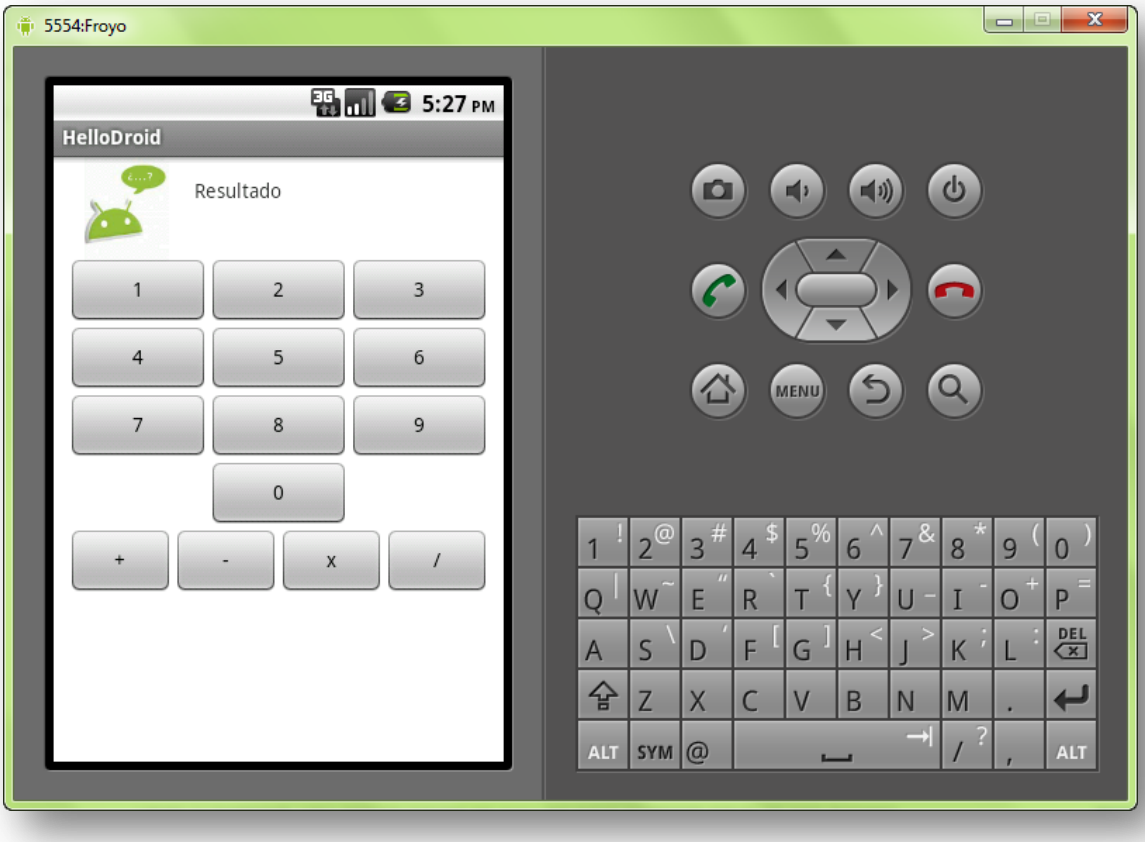

Figura 32. Emulador de Android en ejecución

En el emulador podemos distinguir tres zonas principalmente: la pantalla situada a la izquierda, los botones de acción con los que un dispositivo físico cuenta situados en la parte superior derecha y un teclado virtual que simula los teclados físicos con los que puede contar un dispositivo real, situado en la parte inferior derecha.

En este momento podemos interactuar con la aplicación y comprobar que el comportamiento es el esperado, tal y como se muestra a continuación:

1. Introducir cantidades y operador aritmético.

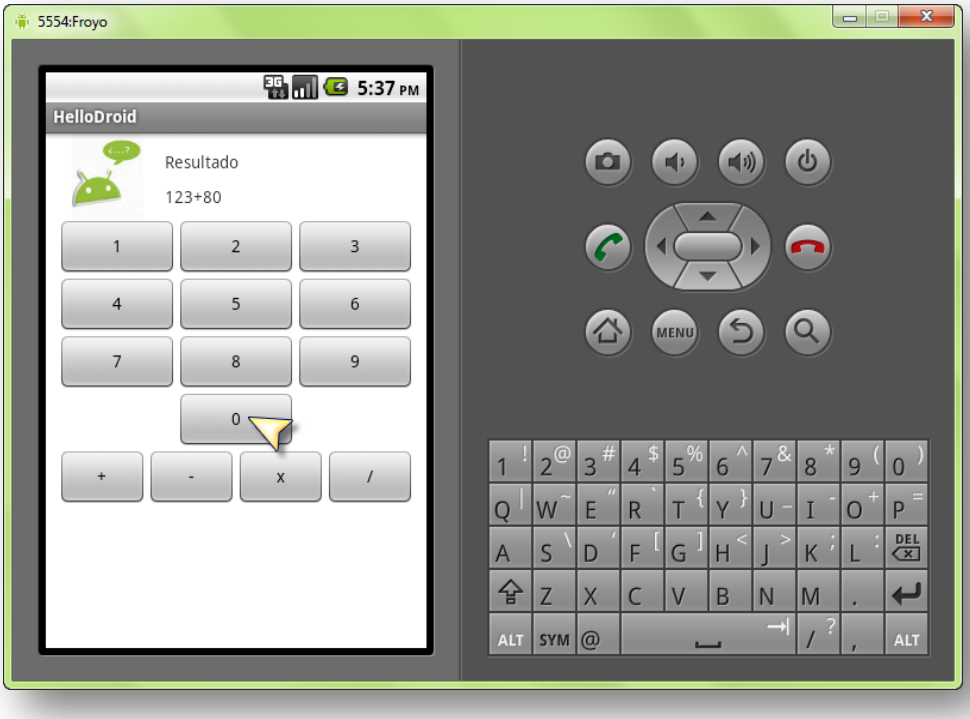

Figura 33. Ejecución de la aplicación (I)

2. Obtener resultado pulsando sobre la imagen del androide.

Figura 34. Ejecución de la aplicación (II)

3. Limpiar el campo de texto donde se muestra el resultado.

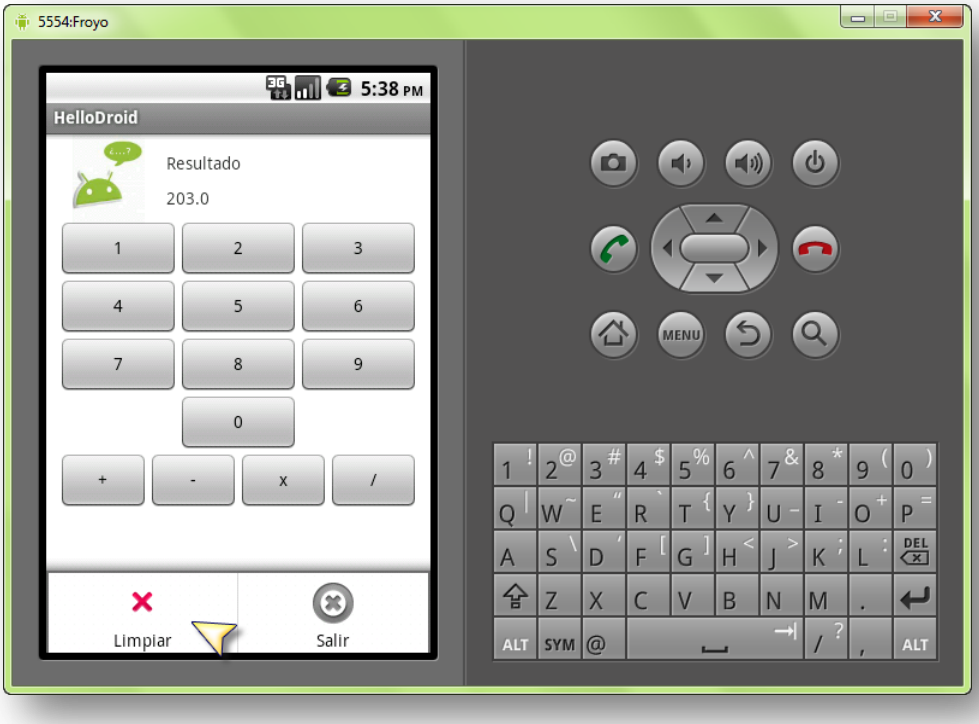

Figura 35. Ejecución de la aplicación (III)

4. Salir de la aplicación, lo que nos devuelve a la pantalla inicial de Android.

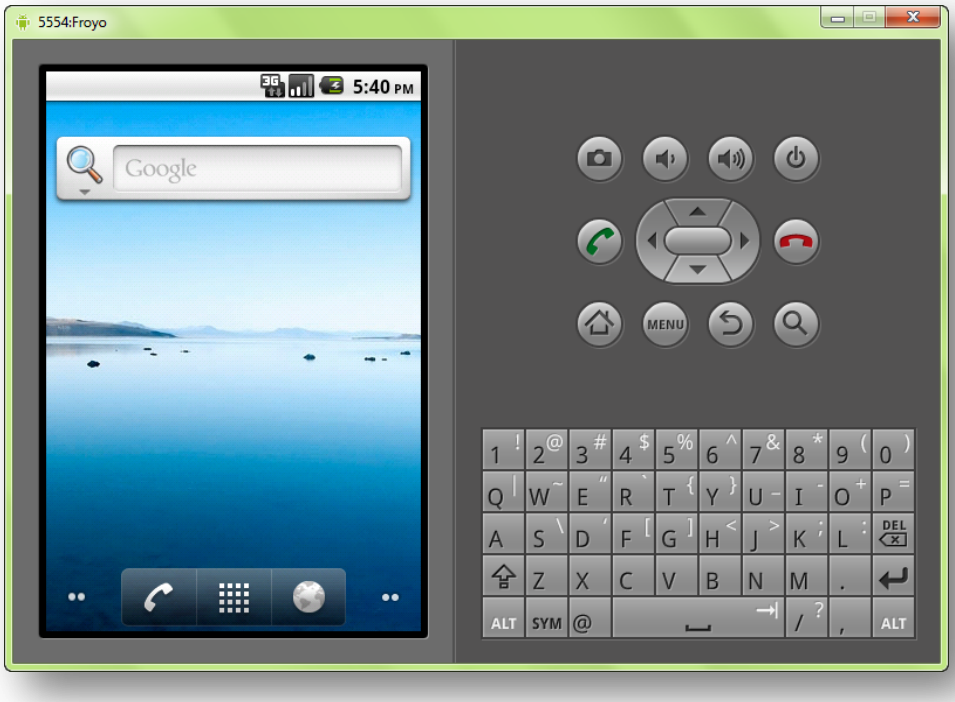

Figura 36. Finalización de la ejecución de la aplicación

Si en lugar de ejecutar la aplicación, iniciamos un proceso de depuración, la aplicación se iniciará en el emulador con el modo de "*Waiting For Debugger*". Esto indica que el depurador está siendo enlazado con el dispositivo virtual, y cuando finalice esta acción se mostrará en Eclipse la vista de depuración en la que se tiene acceso a distintas herramientas [16]muy útiles para la depuración como por ejemplo:

- Visor de procesos e hilos en ejecución.
- Visor de variables y puntos de ruptura.
- Visor de código que permite hacer un seguimiento de por dónde se encuentra la aplicación ejecutando cuando se establecen puntos de ruptura.
- Consola: muestra mensajes acerca de la ejecución de la aplicación y sobre el estado del dispositivo virtual.
- Logcat: monitoriza y muestra todos los eventos que se producen en el emulador (excepciones, memoria liberada, etc.).

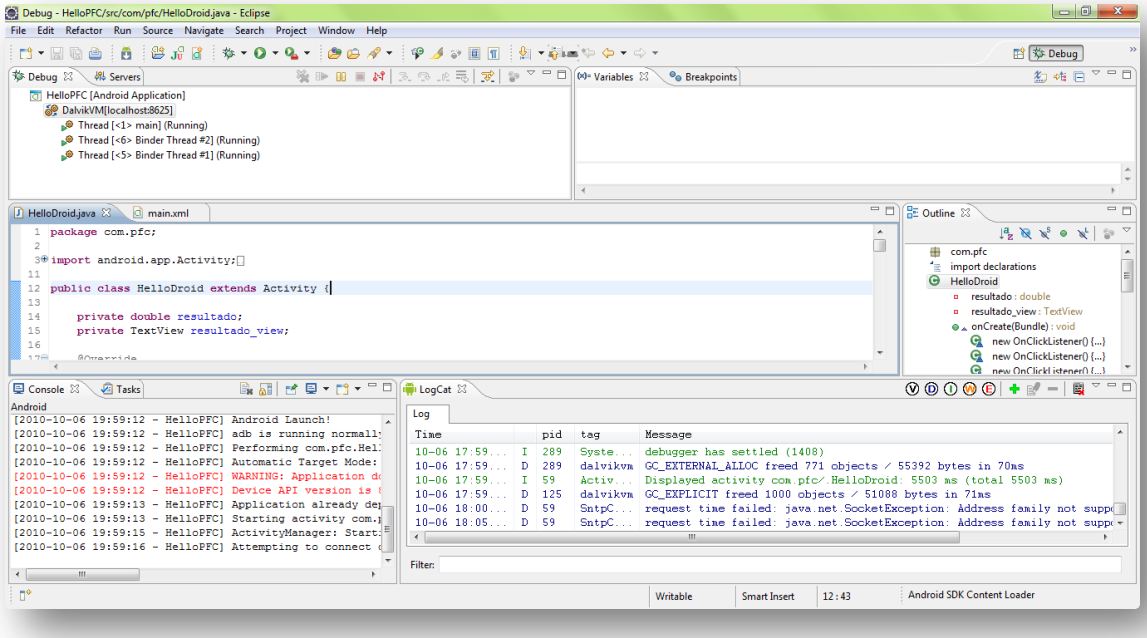

Figura 37. Vista de depuración de Eclipse

Para obtener más información a cerca de todas estas herramientas y otras se recomienda leer la información disponible en el siguiente enlace[16].

Por último, y relacionado con este apartado, en otros puntos se ha indicado la posibilidad de ejecutar las aplicaciones en un dispositivo físico. Para ello es necesario configurar el dispositivo, simplemente activando la opción de "*Depuración USB*" en el menú de "*Desarrollo*" dentro de los ajustes relacionados con aplicaciones del

dispositivo que se utilice, y declarar la aplicación como "*debuggable*" en el archivo *AndroidManifest.xml*. Una vez el dispositivo está conectado por USB al equipo de desarrollo y se han realizado las configuraciones oportunas, solamente queda iniciar la ejecución pulsando el botón ***Run*** (o ***Debug***) de la barra de herramientas de Eclipse.

4.7 Distribuir la aplicación

Cualquier aplicación de Android debe estar firmada digitalmente para que pueda ser instalada en un dispositivo real o en el emulador. Esto se puede hacer de dos formas: con una clave de depuración (válida para ejecutar la aplicación en un emulador o en un dispositivo de depuración) y con una clave privada (válida para distribuir la aplicación).

En el primero de los casos no es necesaria ninguna configuración por parte del usuario pues el plug-in ADT se encarga de firmar digitalmente el paquete APK utilizando una clave de desarrollo generada automáticamente.

En el caso de que se tenga intención de publicar la aplicación en Android Market, se debe realizar el proceso descrito a continuación:

1. Obtener una clave privada que cumpla con los requisitos:
   - Solo está en posesión de la clave la persona o empresa que va a publicar la aplicación.
   - Representa información acerca de la entidad responsable de la aplicación.
   - Tiene una validez que excede el tiempo de vida de la aplicación, siendo recomendable un periodo de 25 años.
   - No se corresponde con la clave de depuración generada por ADT.

   Una forma de obtener una clave de este tipo es utilizando la herramienta *Keytool*, incluida en el SDK de Android, mediante el siguiente comando:

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

Figura 38. Comando utilizado para generar una clave privada válida

2. Una vez generada la clave, podemos exportar la aplicación a un fichero APK firmado y listo para su distribución realizando los siguientes pasos:

a. Desde el menú ***File*** en Eclipse seleccionar ***Export***. En la ventana que aparece en el desplegable de Android escoger ***Export Android Application***.

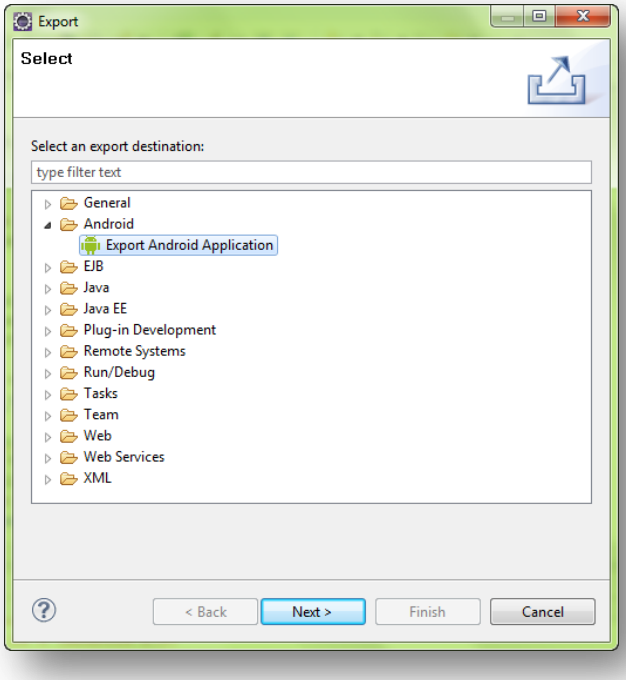

Figura 39. Exportar aplicación (I)

b. En la nueva ventana seleccionar el proyecto que se quiere exportar.

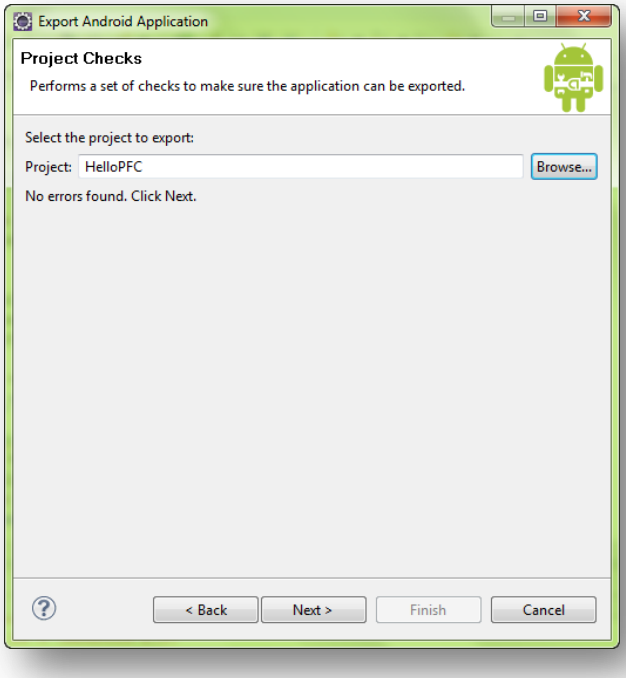

Figura 40. Exportar aplicación (II)

c. Seleccionar el almacén de claves que fue creado anteriormente e introducir la clave de acceso al mismo.

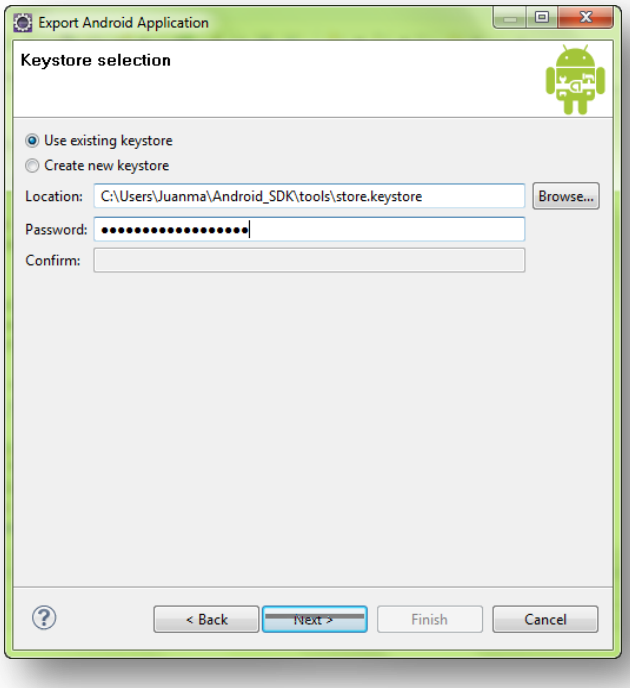

Figura 41. Exportar aplicación (III)

d. Seleccionar la clave privada creada e indicar su clave de acceso.

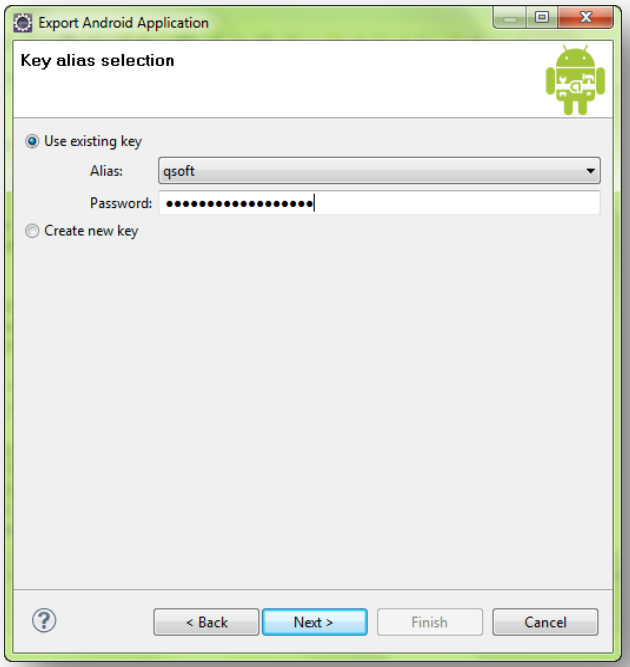

Figura 42. Exportar aplicación (IV)

e. Indicar la ubicación del archivo APK que será generado y pulsar ***Finish***.

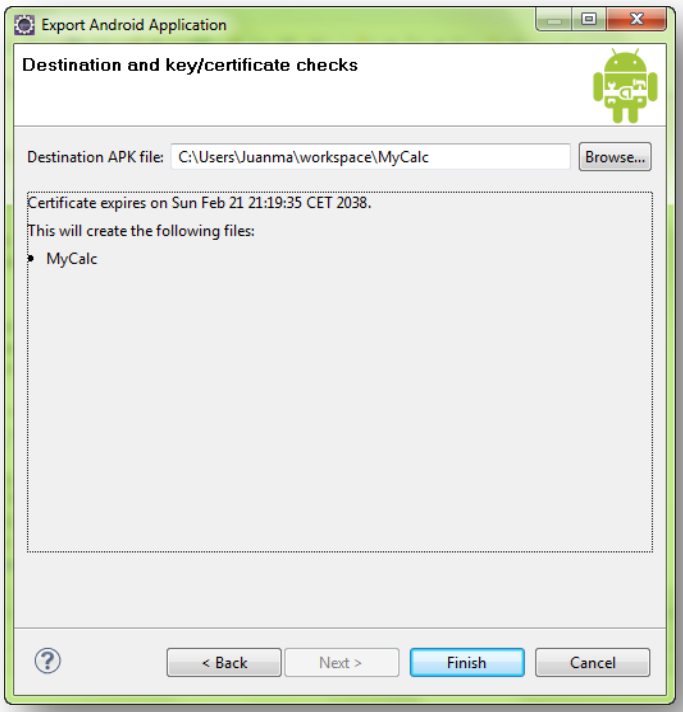

Figura 43. Exportar aplicación (V)

Capitulo 5

Análisis, diseño e implementación de "UC3MSlides"

5.1 Introducción

Una vez asimilados todos los conceptos expuestos en los capítulos tres y cuatro, es momento de pasar a la fase de desarrollo de la aplicación que ha sido planteada para este proyecto.

En primer lugar se realizará un análisis de los requisitos de usuario de la aplicación, estableciendo además varios casos de uso de la misma. A continuación se abordará la fase de diseño del proyecto, con una aproximación a la fase diseño arquitectónico dentro del ciclo de desarrollo de software. Y por último se detallarán los aspectos más relevantes de la implementación de la aplicación.

5.2 Análisis

En este apartado se especificarán los requisitos de usuario además de establecerse varios casos de uso.

5.2.1 Requisitos de usuario

En este apartado se van a describir los distintos requisitos de usuario de la aplicación clasificándolos como requisitos de capacidad o de restricción. Cada requisito está especificado siguiendo un formato tabular conteniendo la siguiente información para cada uno de ellos:

- Identificador: identifica de forma univoca un requisito, siguiendo las reglas de nombrado siguientes: CAP-X, para los requisitos de capacidad, y RES-X, para los requisitos de restricción. El valor de X es numérico empezando en uno.
- Descripción: Especificación detallada del requisito.
- Necesidad: Establece la necesidad del requisito dentro del proyecto, toma un valor dentro de la escala 1-4, siendo 4 la necesidad más alta y 1 la más baja.
- Prioridad: Establece la prioridad del requisito dentro del proyecto, toma un valor dentro de la escala 1-4, siendo 4 la prioridad más alta y 1 la más baja.
- Estabilidad: Establece la sensibilidad del requisito a ser modificado, toma los valores de “Estable” o “No Estable”.

5.2.1.1 Requisitos de capacidad

A continuación en este apartado se muestran los requisitos de usuario de capacidad:

Identificador	**CAP_1: Listar ficheros**
Descripción	El usuario será capaz de ver los contenidos de su tarjeta de memoria en forma de listado.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 7. Requisito de capacidad #1

Identificador	**CAP_2: Navegar por directorios**
Descripción	El usuario será capaz de navegar por los directorios de su tarjeta de memoria.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 8. Requisito de capacidad #2

Identificador	**CAP_3: Borrar elementos**
Descripción	El usuario podrá eliminar los ficheros o directorios que contenga la tarjeta de memoria.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 9. Requisito de capacidad #3

Identificador	**CAP_4: Copiar elementos**
Descripción	El usuario podrá copiar elementos de una ubicación de la tarjeta de memoria para después pegarlos en otra ubicación distinta.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 10. Requisito de capacidad #4

Identificador	**CAP_5: Mover elementos**
Descripción	El usuario podrá mover elementos de una ubicación a otra en la estructura de directorios de su tarjeta de memoria.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 11. Requisito de capacidad #5

Identificador	**CAP_6: Pegar elementos copiados**
Descripción	EL usuario puede pegar los elementos que previamente haya copiado en otra ubicación de su tarjeta de memoria.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 12. Requisito de capacidad #6

Identificador	**CAP_7: Renombrar elementos**
Descripción	El usuario puede modificar el nombre de los elementos que contenga su tarjeta, ya sean ficheros o directorios.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 13. Requisito de capacidad #7

Identificador	**CAP_8: Crear directorios**
Descripción	El usuario puede crear nuevos directorios dentro de la tarjeta de memoria.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 14. Requisito de capacidad #8

Identificador	**CAP_9: Consultar propiedades**
Descripción	El usuario puede consultar las propiedades de un archivo o directorio presentes en la tarjeta de almacenamiento.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 15. Requisito de capacidad #9

Identificador	**CAP_10: Idioma**
Descripción	La aplicación se podrá mostrar en varios idiomas, como mínimo inglés y español.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 16. Requisito de capacidad #10

Identificador	**CAP_11: Configuración**
Descripción	El usuario podrá configurar varios aspectos de la aplicación como son los siguientes: • La dirección IP de la máquina que actúa de servidor y a la cual se va conectar. • La carpeta en la que se guardan los archivos que recibe desde el mismo. • Borrar el historial de transferencias.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 17. Requisito de capacidad #11

Identificador	**CAP_12: Asociar alias**
Descripción	El usuario podrá asociar un alias a la dirección IP del servidor para reconocer más fácilmente el mismo.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 18. Requisito de capacidad #12

Identificador	**CAP_13: Guardar varias direcciones de red**
Descripción	El usuario puede almacenar varias direcciones de red cada una de ellas asociada con un alias o ubicación, de manera que pueda cambiar de servidor de forma sencilla.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 19. Requisito de capacidad #13

Identificador	**CAP_14: Conectividad**
Descripción	El usuario puede configurar los parámetros de la red Wifi o de datos 3G accediendo a los ajustes del teléfono a través de las opciones de configuración de la aplicación, pudiendo habilitar la conectividad o añadir nuevos puntos de acceso.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 20. Requisito de capacidad #14

Identificador	**CAP_15: Transferir archivos**
Descripción	El usuario podrá transferir archivos a un equipo que actúe de servidor para por ejemplo mantener una copia de seguridad de los mismos.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 21. Requisito de capacidad #15

Identificador	**CAP_16: Recibir archivos**
Descripción	El usuario podrá recibir archivos desde el equipo servidor.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 22. Requisito de capacidad #16

Identificador	**CAP_17: Consultar historial**
Descripción	La aplicación permitirá al usuario consultar un historial de las transferencias que se han realizado entre cliente y servidor.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 23. Requisito de capacidad #17

Identificador	**CAP_18: Eliminar historial**
Descripción	El usuario podrá eliminar el historial de las transferencias entre el teléfono y el equipo servidor accediendo al menú de configuración de la aplicación.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 24. Requisito de capacidad #18

Identificador	**CAP_19: Filtrar archivos**
Descripción	Se podrán filtras los archivos que contiene la tarjeta de almacenamiento en base al tipo de archivo, mostrándose únicamente los archivos que cumplan el criterio establecido en el filtro. Los tipos de archivo que pueden filtrarse son: • Imágenes: gif, bmp, jpg y png. • Música: wma, wav y mp3. • Video: avi, wmv, mp4 y 3gp. • Portable document format o PDF. • Documentos de office: ppt, pptx, doc, docx.
Necesidad	3
Prioridad	3
Estabilidad	Estable

Tabla 25. Requisito de capacidad #19

Identificador	**CAP_20: Abrir archivos**
Descripción	La aplicación permitirá al usuario ejecutar otras actividades del sistema para abrir ciertos tipos de archivo como por ejemplo: • Imágenes • Videos • Música
Necesidad	2
Prioridad	2
Estabilidad	Estable

Tabla 26. Requisito de capacidad #20

Identificador	**CAP_21: Crear presentaciones de imágenes**
Descripción	El usuario podrá crear presentaciones de imágenes seleccionando de entre las que tenga en su teléfono varias de ellas para mandarlas el equipo servidor donde de forma automática se abrirá el visor predeterminado del sistema operativo en el que se ejecuta.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 27. Requisito de capacidad #21

Identificador	**CAP_22: Enviar presentaciones**
Descripción	El usuario podrá enviar archivos en formato PDF o de PowerPoint al servidor donde se abrirá el visor predeterminado del sistema operativo en el que se está ejecutando.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 28. Requisito de capacidad #22

Identificador	**CAP_23: Controlar presentaciones**
Descripción	El usuario podrá controlar las presentaciones que haya enviado al equipo servidor mediante la interfaz mostrada en la pantalla del terminal móvil. Pudiendo cambiar de diapositiva o imagen o poner la presentación en pantalla completa.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 29. Requisito de capacidad #23

Identificador	**CAP_24: Obtener diapositivas**
Descripción	El usuario podrá obtener la diapositiva que se esté mostrando en ese momento en la pantalla conectada al equipo servidor descargándola en su teléfono móvil.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 30. Requisito de capacidad #24

Identificador	**CAP_25: Crear anotaciones**
Descripción	La aplicación permitirá asociar anotaciones con las diapositivas obtenidas del servidor.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 31. Requisito de capacidad #25

Identificador	**CAP_26: Ver anotaciones**
Descripción	El usuario podrá ver las anotaciones creadas para las distintas diapositivas que haya capturado de una presentación.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 32. Requisito de capacidad #26

Identificador	**CAP_27: Eliminar anotaciones**
Descripción	El usuario tendrá la posibilidad de eliminar las anotaciones escritas.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 33. Requisito de capacidad #27

Identificador	**CAP_28: Varios usuarios**
Descripción	El servidor será capaz de atender varios usuarios a la vez de forma que uno controle la presentación y el resto pueda obtener las diapositivas de la misma.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 34. Requisito de capacidad #28

Identificador	**CAP_29: Directorio público**
Descripción	El usuario podrá obtener de forma remota archivos contenidos en un directorio público dentro del servidor, sin necesidad de que el usuario tenga que interactuar con la interfaz del servidor para iniciar la transferencia.
Necesidad	4
Prioridad	3
Estabilidad	Estable

Tabla 35. Requisito de capacidad #29

5.2.1.2 Requisitos de restricción

A continuación se encuentras especificados los requisitos de usuario de restricción:

Identificador	**RES_1: Plataforma Android**
Descripción	El dispositivo en el que se instale la aplicación debe contar con una versión de Android igual o superior a la 2.2.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 36. Requisito de restricción #1

Identificador	**RES_2: Permisos**
Descripción	Al instalar la aplicación el usuario debe otorgar los permisos necesarios a la aplicación para que esta pueda funcionar de forma correcta en el teléfono. Estos permisos son: • Montar la unidad de almacenamiento externo. • Escribir en la unidad de almacenamiento externo. • Acceso a internet. • Consultar el estado de interfaz Wifi. • Consultar el estado de la interfaz de datos 3G.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 37. Requisito de restricción #2

Identificador	**RES_3: Java JRE**
Descripción	La aplicación servidora debe instalarse en un equipo que cuente con un JRE con versión igual o superior a la 1.6.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 38. Requisito de restricción #3

Identificador	**RES_4: Formato de direcciones IP**
Descripción	Las direcciones IP que se consideran válidas para la aplicación deberán seguir el formato W.X.Y.Z donde estos parámetros deben ser números entre el 0 y el 255.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 39. Requisito de restricción #4

Identificador	**RES_5: Puertos habilitados**
Descripción	El equipo servidor deberá tener habilitados y libres los puertos 4443, 4444 y 4445 para la correcta comunicación con los dispositivos clientes.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 40. Requisito de restricción #5

Identificador	**RES_6: Idioma de la interfaz**
Descripción	La interfaz mostrará por defecto sus contenidos en inglés, mostrándose en español en el caso de que en el terminal las opciones de localización estén configuradas en este idioma.
Necesidad	3
Prioridad	3
Estabilidad	Estable

Tabla 41. Requisito de restricción #6

Identificador	**RES_7: Acceso a protegido al servidor**
Descripción	El servidor deberá solicitar una contraseña de acceso para el dispositivo móvil de forma que este pueda enviar archivos o controlar las presentaciones.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 42. Requisito de restricción #7

Identificador	**RES_8: Acceso protegido a la carpeta publica**
Descripción	El usuario deberá proveer una contraseña de acceso para poder obtener los archivos alojados en el directorio público del servidor.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 43. Requisito de restricción #8

Identificador	**RES_9: Términos de uso**
Descripción	Cuando se inicie por primera vez la aplicación el usuario deberá aceptar la licencia de uso.
Necesidad	1
Prioridad	1
Estabilidad	Estable

Tabla 44. Requisito de restricción #9

Identificador	**RES_10: Acceso a otras particiones de datos**
Descripción	El usuario no podrá ver los contenidos de otras particiones de datos del dispositivo móvil que no sea la de almacenamiento externo.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 45. Requisito de restricción #10

Identificador	**RES_11: Conectividad**
Descripción	Deberá existir conectividad Wifi o 3G para poder acceder a todas las funcionalidades de la aplicación.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 46. Requisito de restricción #11

Identificador	**RES_12: Portabilidad**
Descripción	El programa servidor deberá funcionar correctamente en las distintas plataformas para las que está pensado: Windows, Linux y Mac.
Necesidad	4
Prioridad	4
Estabilidad	Estable

Tabla 47. Requisito de restricción #12

5.2.1.3 Casos de uso

Durante este apartado se especificarán los diferentes casos de uso planteados para la aplicación que va a ser desarrollada, teniendo en cuenta que la misma tiene a la vez un carácter servidor y cliente, y por lo tanto, el usuario podrá interactuar de forma distinta.

Cada caso de uso va a ser especificado de forma gráfica con un diagrama que muestre la interacción del usuario con el sistema, y además se detallará los pasos necesarios para cada escenario con un formato tabular descrito a continuación:

- Identificador: identifica de forma univoca un caso de uso siguiendo el formato de nombrado CU_X, donde X se corresponde a un número empezando en la unidad.
- Descripción: describe los pasos realizados por el usuario para la situación planteada.
- Pre-condiciones: condiciones que deben darse para la realización del caso de uso.
- Post-condiciones: condiciones que son resultado de la ejecución del caso de uso.

Identificador	**CU_1: Manipular archivos y directorios en el dispositivo móvil**
Descripción	• El usuario inicia la aplicación. • El usuario puede navegar por los directorios presionando sobre los elementos del listado que aparece. • Al seleccionar uno o varios de los elementos, puede abrir el menú de edición y realizar una de las acciones disponibles: Crear directorio, Borrar, Copiar, Mover y Pegar • Si pulsa durante varios segundos en uno de los elementos podrá realizar otras acciones: Ver las propiedades o renombrar el elemento.
Pre-condiciones	• La tarjeta de memoria está insertada en el dispositivo. • La tarjeta está montada en el sistema.
Post-condiciones	• El usuario a modificado un elemento

Tabla 48. Caso de uso #1

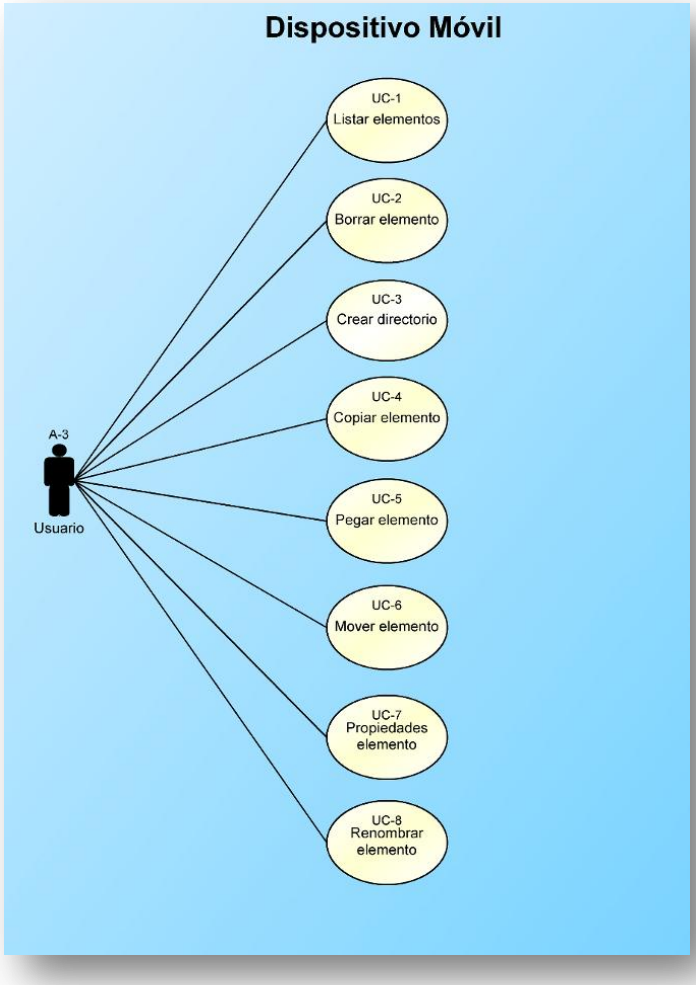

Figura 44: Caso de uso #1

Identificador	**CU_2: Enviar archivos al servidor**
Descripción	• Explorar los directorios en busca de los archivos que se quieren transferir. • Seleccionar el/los archivos. • Desde el menú de la aplicación seleccionar “Transferir”.
Pre-condiciones	• Tarjeta de memoria disponible. • Conexión a la red disponible. • Aplicación configurada.
Post-condiciones	• Los archivos son enviados al servidor.

Tabla 49. Caso de uso #2

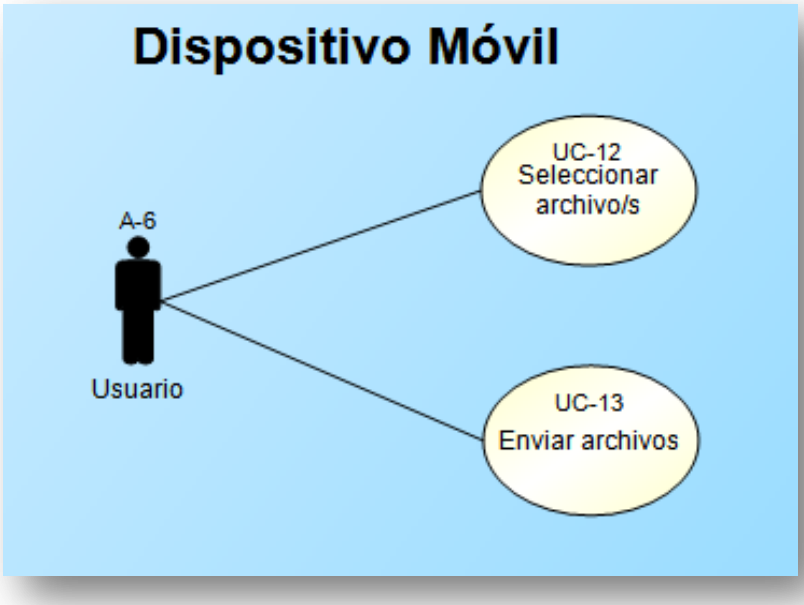

Figura 45: Caso de uso #2

Identificador	**CU_3: Realizar presentación de imágenes**
Descripción	• Explorar los directorios en busca de las imágenes que se quieren transferir para realizar la presentación. • Seleccionar las imágenes. • En el menú de la aplicación seleccionar “Presentación” y elegir la opción “Imágenes”. • Una vez que se han transferido las imágenes el usuario puede controlar la presentación desde la interfaz presentada en el dispositivo móvil pudiendo avanzar o retroceder la imagen que se muestra o ver una presentación en pantalla completa. • Terminar presentación.
Pre-condiciones	• Tarjeta de memoria disponible. • Conexión a la red disponible. • Aplicación configurada.
Post-condiciones	• Se muestra una presentación de las imágenes seleccionadas.

Tabla 50. Caso de uso #3

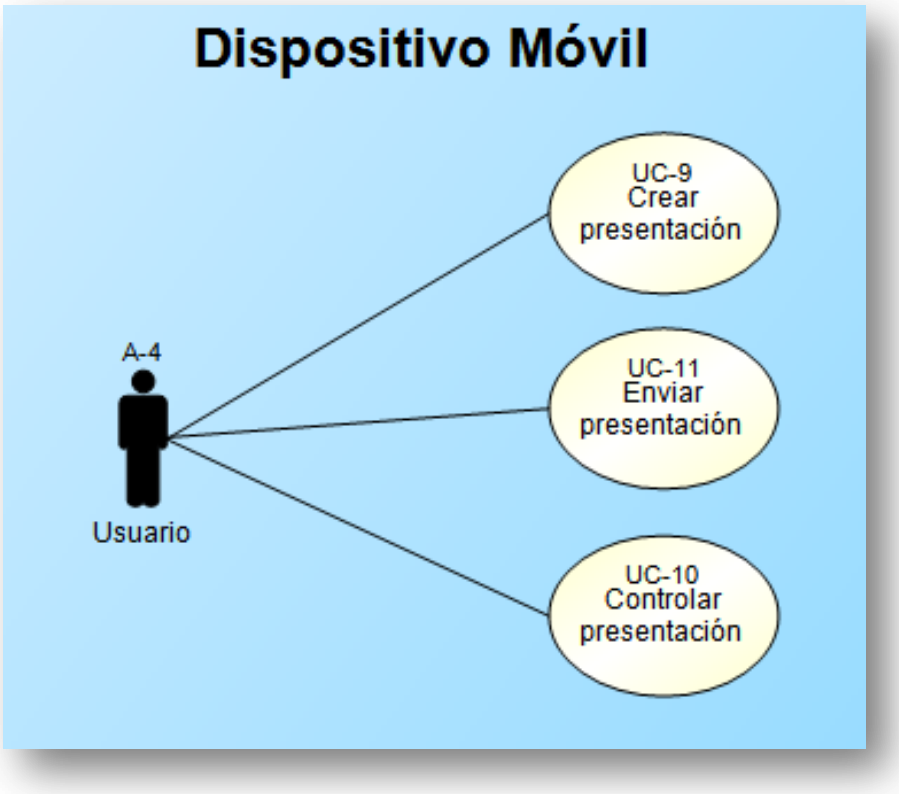

Figura 46: Caso de uso #3

Identificador	**CU_4: Realizar presentación de diapositivas**
Descripción	• Explorar los directorios en busca de la presentación en PDF o PowerPoint que se quieren transferir. • Seleccionar el archivo. • En el menú de la aplicación seleccionar “Presentación” y elegir la opción “PDF o PowerPoint”. • Una vez que se han transferido el archivo el usuario puede controlar la presentación desde la interfaz presentada en el dispositivo móvil pudiendo avanzar o retroceder la diapositiva que se muestra o ver una presentación en pantalla completa. • Terminar presentación.
Pre-condiciones	• Tarjeta de memoria disponible. • Conexión a la red disponible. • Aplicación configurada.
Post-condiciones	• Se muestra una presentación de diapositivas.

Tabla 51. Caso de uso #4

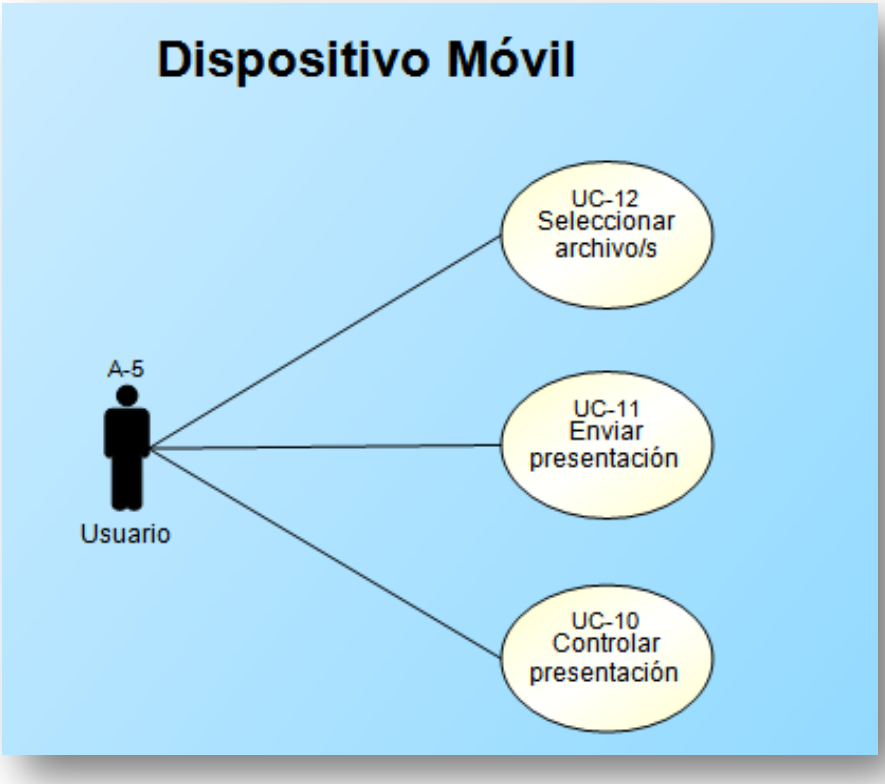

Figura 47: Caso de uso #4

Identificador	**CU_5: Añadir elementos al directorio publico**
Descripción	• Desde la interfaz del servidor, seleccionar el botón “Publicar”. • En el diálogo que se abre seleccionar los archivos que se quieren añadir al directorio público del servidor. • Pulsar “Aceptar”. • Automáticamente se añaden los archivos al directorio público.
Pre-condiciones	---
Post-condiciones	• El cliente puede descargar los elementos que almacena este directorio.

Tabla 52. Caso de uso #5

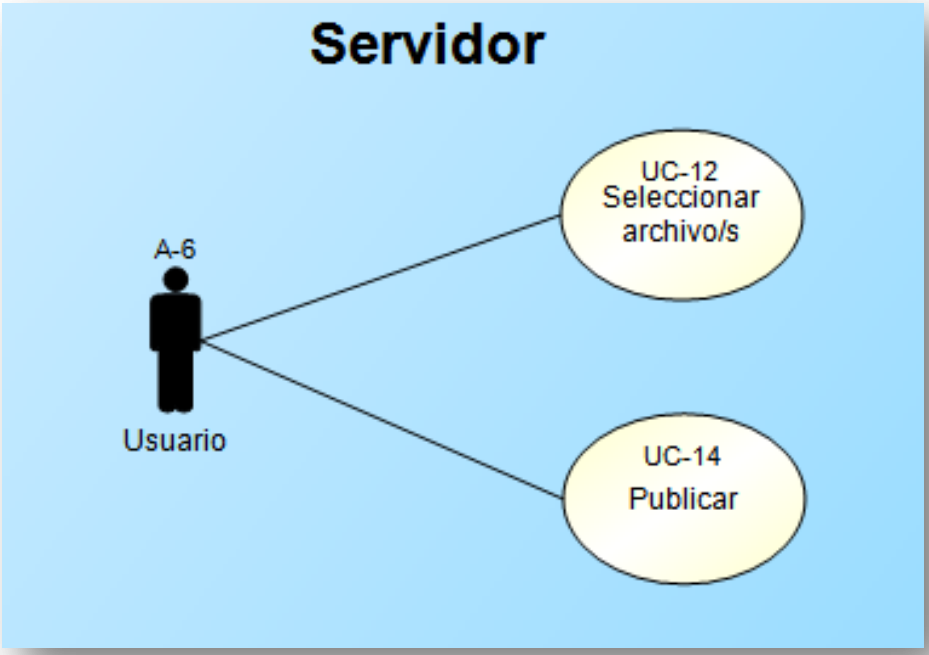

Figura 48: Caso de uso #5

Identificador	**CU_6: Obtener elementos del directorio público**
Descripción	• Seleccionar en el menú de la aplicación “Obtener archivos”. • Indicar la contraseña del directorio público. • En el listado mostrado seleccionar el/los archivos que queramos descargar del servidor. • Automáticamente se empezarán a descargar estos archivos.
Pre-condiciones	• Tarjeta de memoria disponible. • Conexión a la red disponible. • Aplicación configurada.
Post-condiciones	• Se obtienen el/los ficheros seleccionados.

Tabla 53. Caso de uso #6

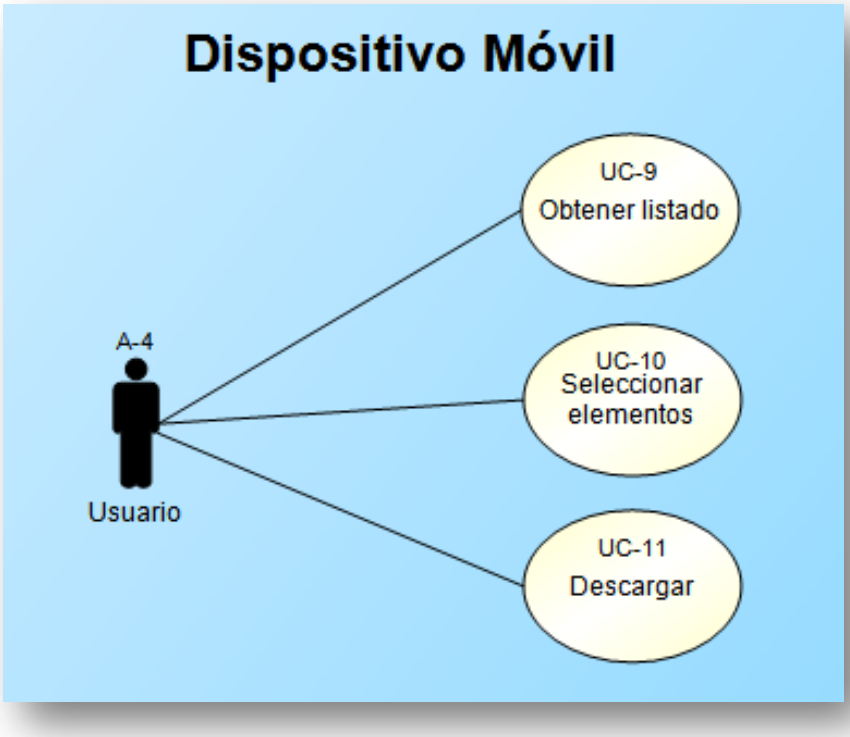

Figura 49: Caso de uso #6

Identificador	**CU_7: Filtrar archivos**
Descripción	• En el menú de la aplicación seleccionar “Filtrar”. • De entre las opciones mostradas elegir el filtro que más nos interese.
Pre-condiciones	• Tarjeta de memoria disponible.
Post-condiciones	• Se muestra un listado de archivos que cumplen el criterio especificado.

Tabla 54. Caso de uso #7

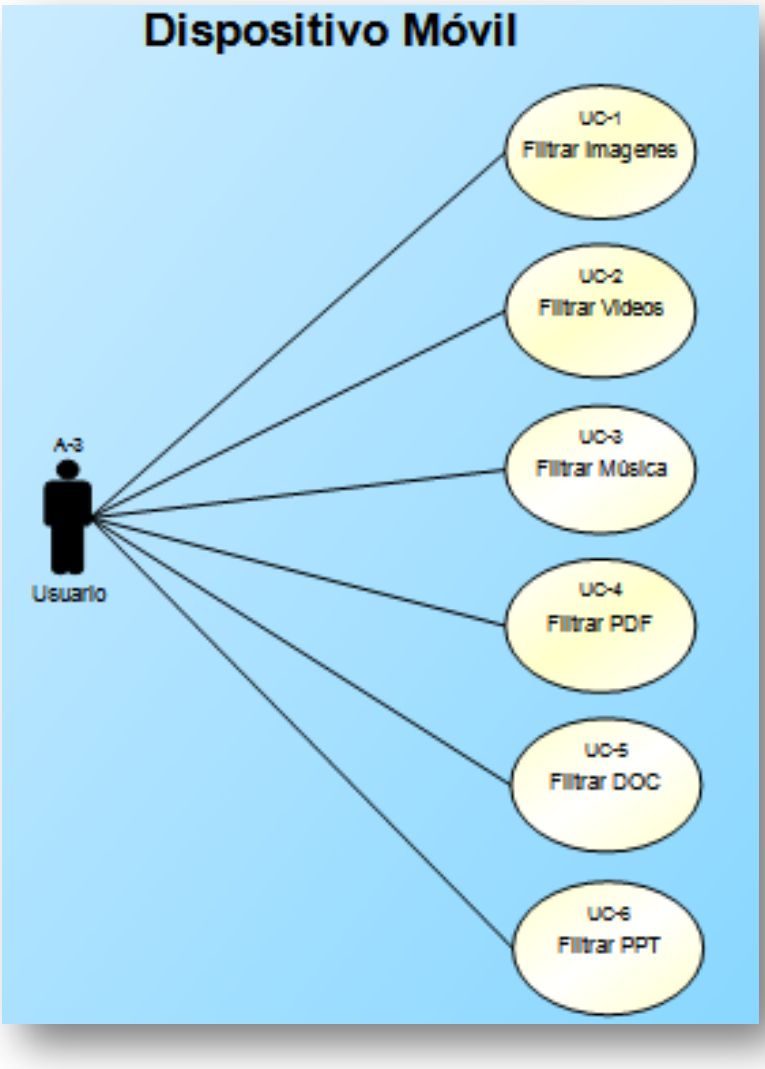

Figura 50: Caso de uso #7

Identificador	**CU_8: Configurar aplicación**
Descripción	• En el menú de la aplicación seleccionar “Ajustes”. • De entre los parámetros de configuración mostrados elegir uno y cambiarlo, pudiéndose añadir una nueva dirección de red, cambiar el directorio de descargar, etc.
Pre-condiciones	---
Post-condiciones	• La aplicación queda configurada por el usuario.

Tabla 55. Caso de uso #8

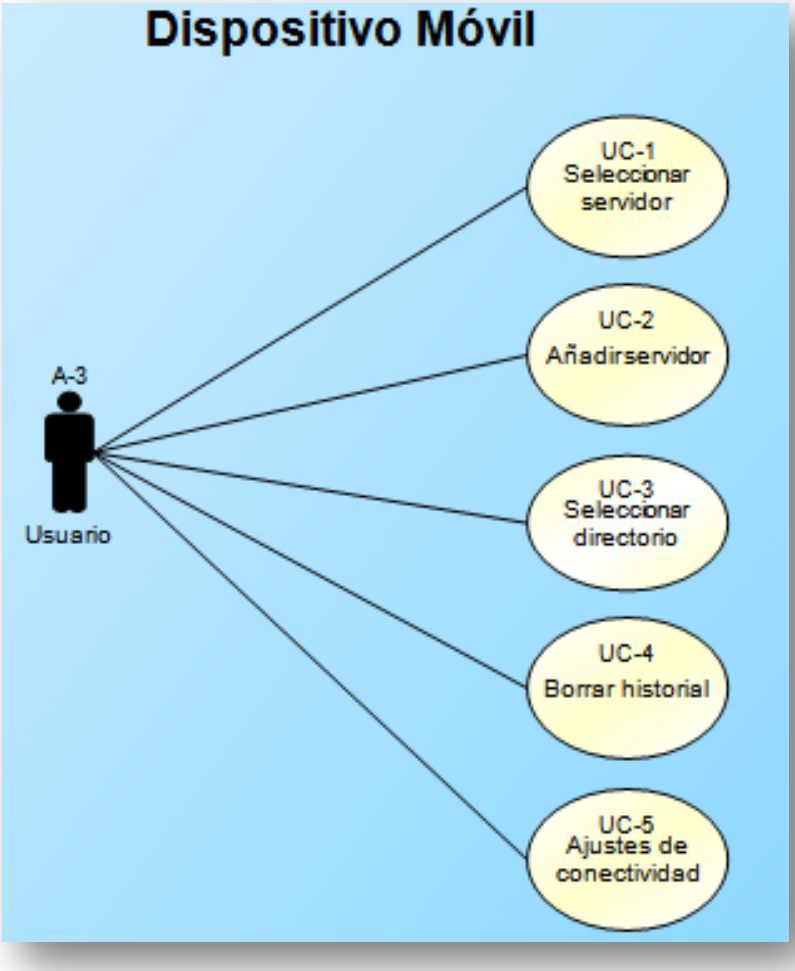

Figura 51: Caso de uso #8

Identificador	**CU_9: Obtener diapositivas y añadir anotaciones**
Descripción	• En el menú de la aplicación seleccionar "Presentación" sin haber seleccionado ningún archivo. • En la interfaz se muestra un botón con el que obtener la diapositiva que se esté mostrando. • Cuando se descargue la imagen se abrirá un editor de texto en el que realizar la anotación. • Guardar la anotación, tras ello se mostrará la imagen y la anotación juntas.
Pre-condiciones	• Tarjeta de memoria disponible. • Conexión a la red disponible. • Aplicación configurada.
Post-condiciones	• Se crea una anotación para una diapositiva en particular

Tabla 56. Caso de uso #9

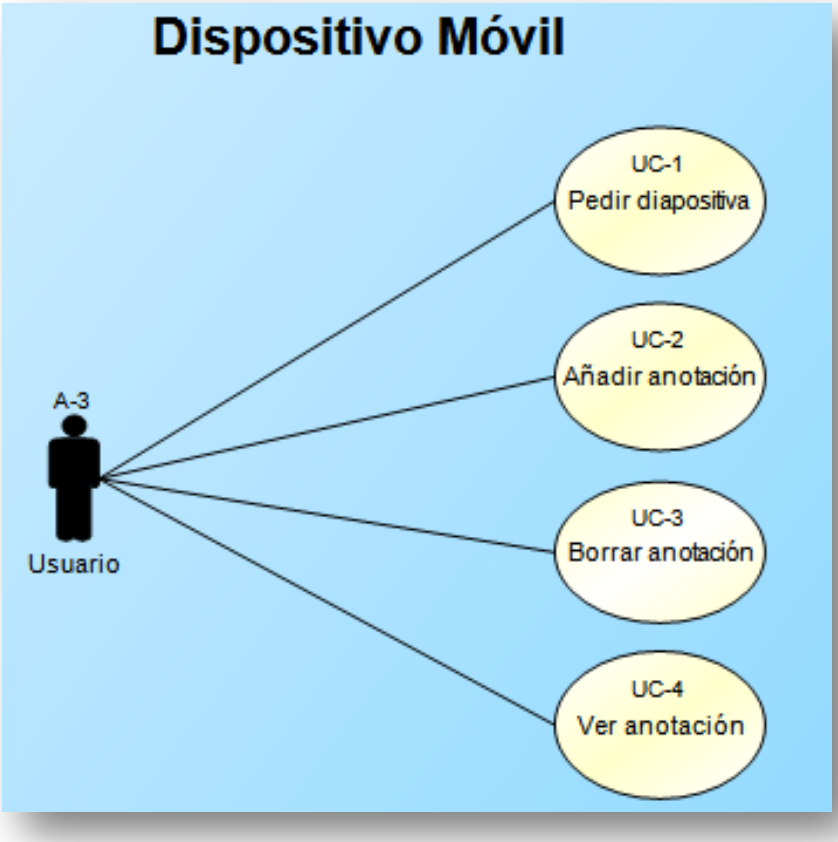

Figura 52: Caso de uso #9

Identificador	**CU_10: Abrir archivos**
Descripción	• Navegar por los contenidos del dispositivo de almacenamiento externo. • Seleccionar un archivo. • Al hacerlo se abrirá un visor o reproductor para un archivo de música, video o imágenes.
Pre-condiciones	• Tarjeta de memoria disponible.
Post-condiciones	• Se muestra o reproduce el archivo seleccionado

Tabla 57. Caso de uso #10

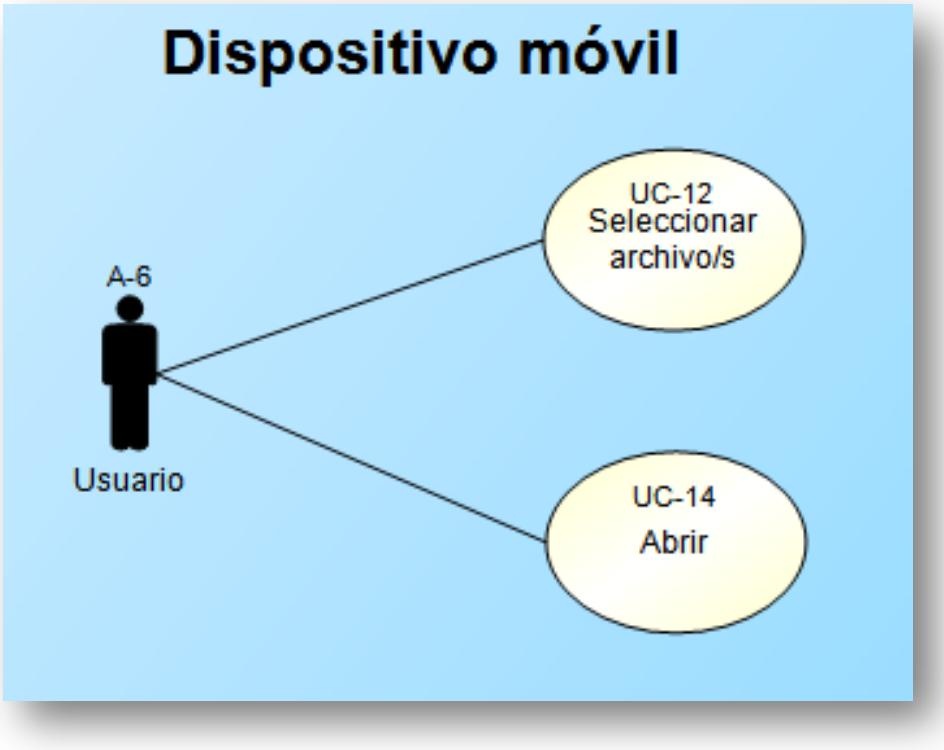

Figura 53: Caso de uso #10

5.3 Diseño

Durante esta sección se discutirán los aspectos relacionados con la fase de diseño dentro de ciclo de desarrollo de software. Concretamente se van a comentar los detalles relacionados con el diseño arquitectónico del sistema, el diseño de las interfaces de usuario y el diseño de la base de datos.

5.3.1 Diseño arquitectónico

La aplicación desarrollada se basa en una arquitectura Cliente-Servidor donde una entidad cliente realiza peticiones a otra entidad servidora y esta le devuelve una respuesta.

En este caso particular, la entidad cliente será el dispositivo móvil que ejecuta la aplicación de Android y que realizará las peticiones a la aplicación servidora a través de la red, ya sea local o por internet.

En la siguiente figura se muestra un esquema de la arquitectura propuesta identificando las entidades cliente y servidor y mostrando un ejemplo de comunicación entre las mismas:

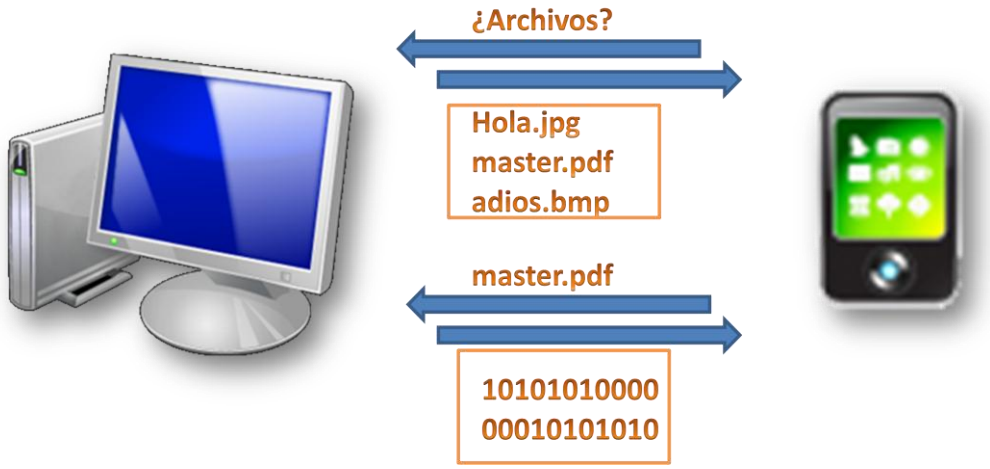

Figura 54: Diagrama explicativo de la arquitectura

En la siguiente figura están representados los principales componentes de la aplicación cliente del sistema desarrollado así como las relaciones que existen entre los mismos.

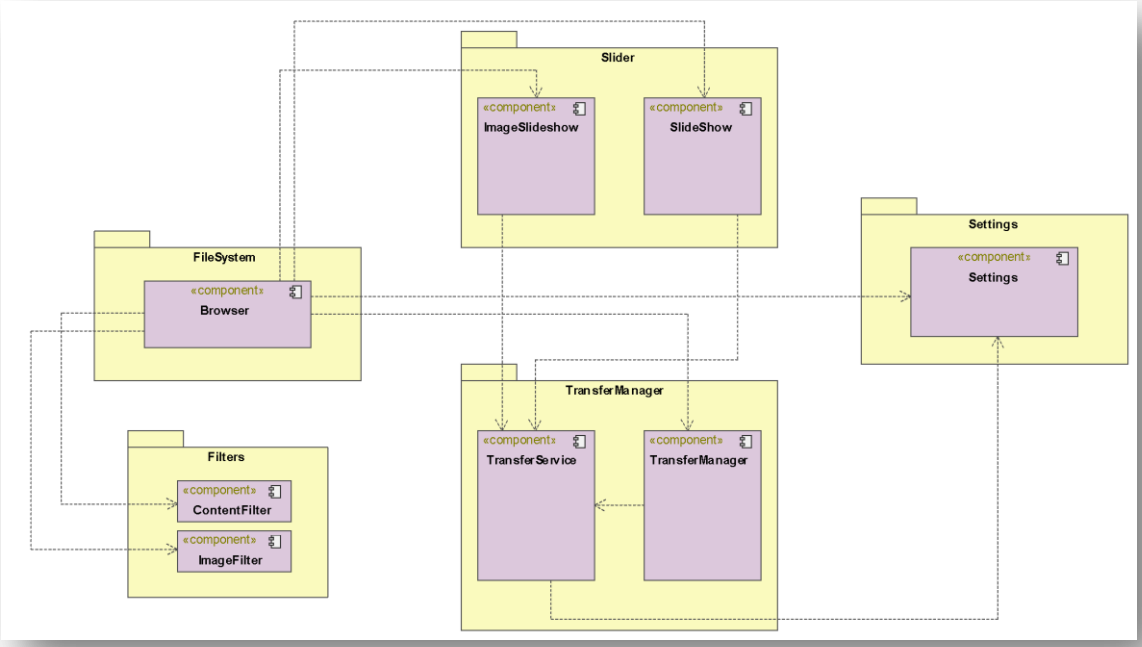

Figura 55: Diagrama de componentes

Para cada uno de los componentes mostrados en la figura anterior, más el componente servidor, se va a incluir una especificación de todos ellos siguiendo un formato tabular que contendrá la siguiente información:

- Identificador: Identifica de forma univoca un componente del sistema siguiendo el formato COMP_X, donde X se debe sustituir por un número empezando en la unidad. Además del identificador se indica también un nombre del componente.
- Propósito: indica por qué se ha diseñado este componente.
- Funciones: lista las funciones del componente dentro del sistema.
- Referencias: especifica de donde surge este componente en relación a los requisitos de usuario descritos en la fase de análisis.

Identificador	**COMP_1: Explorador**
Propósito	Este componente del sistema permite listar los elementos almacenados en el dispositivo de memoria y realizar operaciones sobre los mismos como crear carpetas, borrar elementos, copiar, cortar o pegar. Este componente es el núcleo de la aplicación cliente pues desde él se puede interactuar con el resto de componentes.
Funciones	• Listar. • Crear carpeta. • Copiar. • Cortar. • Pegar. • Renombrar. • Ver propiedades. • Abrir archivos.
Referencias	CAP_1, CAP_2, CAP_3, CAP_4, CAP_5, CAP_6, CAP_7, CAP_8, CAP_9, CAP_20, RES_1, RES,2, RES_6, RES_9, RES_10, RES_11

Tabla 58. Especificación del componente COMP_1

Identificador	**COMP_2: Ajustes**
Propósito	Gestionar la configuración del la aplicación cliente.
Funciones	• Seleccionar servidor. • Añadir direcciones de red. • Comprobar formato de direcciones de red. • Seleccionar carpeta de descargas. • Borrar historial de transferencias. • Acceso a configuración de red del dispositivo.
Referencias	CAP_10, CAP_11, CAP_12, CAP_13, CAP_14, RES_1, RES_2, RES_6, RES_9

Tabla 59. Especificación del componente COMP_2

Identificador	**COMP_3: Filtro contenido**
Propósito	Filtrar los archivos del sistema mostrando solo los que cumplen la condición de filtrado.

Funciones	• Filtro por tipo de contenido:
	  • Video.
	  • Música.
	  • PDF.
	  • Documentos Office (doc, ppt).

Referencias	CAP_19, RES_1, RES_2, RES_6, RES_9

Tabla 60. Especificación del componente COMP_3

Identificador	**COMP_4: Filtro imágenes**
Propósito	Filtrar las imágenes que se encuentran en el dispositivo de almacenamiento.
Funciones	• Filtro de imágenes.
Referencias	CAP_19, RES_1, RES_2, RES_6, RES_9

Tabla 61. Especificación del componente COMP_4

Identificador	**COMP_5: Gestor de transferencias**
Propósito	Registrar todas las transferencias de archivos realizadas entre cliente y servidor.
Funciones	• Mantener historial de transferencias.
Referencias	CAP_17, CAP_18, RES_1, RES_2, RES_6, RES_9

Tabla 62. Especificación del componente COMP_5

Identificador	**COMP_6: Servicio de transferencia**
Propósito	Gestionar el envío y recepción de datos entre cliente y servidor.
Funciones	• Recibir archivos • Enviar archivos
Referencias	CAP_15, CAP_16, RES_1, RES_2, RES_4, RES_5, RES_6, RES_9

Tabla 63. Especificación del componente COMP_6

Identificador	**COMP_7: Presentación**
Propósito	Controlar presentación de forma remota para archivos de PowerPoint y PDF.

Funciones	• Controlar presentaciones:
	  • Siguiente diapositiva
	  • Anterior diapositiva
	  • Pantalla completa
	  • Terminar
	• Obtener presentaciones y gestionar anotaciones

Referencias	CAP_22, CAP_24, CAP_25, CAP_26, CAP_27, CAP_28, RES_1, RES_2, RES_6, RES_9

Tabla 64. Especificación del componente COMP_7

Identificador	**COMP_8: Presentación de imágenes**
Propósito	Controlar las presentaciones de imágenes.

Funciones	• Controlar presentaciones:
	  • Siguiente diapositiva
	  • Anterior diapositiva
	  • Pantalla completa
	  • Terminar

Referencias	CAP_21, CAP_23, RES_1, RES_2, RES_6, RES_9

Tabla 65. Especificación del componente COMP_8

Los componentes anteriores están todos referidos a la aplicación cliente, no obstante, faltaría por especificar el propósito y funciones de los componentes del servidor. En la tabla a continuación se detalla el único componente diseñado para la aplicación servidor:

Identificador	**COMP_9: Servidor**
Propósito	Atender y resolver todas las peticiones que provengan de la aplicación cliente.
Funciones	• Enviar archivos al cliente. • Iniciar aplicaciones por defecto para realizar las presentaciones. • Ejecutar acciones de control sobre las presentaciones. • Gestionar elementos del directorio público. • Crear y gestionar un log de los eventos ocurridos en el sistema.
Referencias	CAP_15, CAP_16, CAP_29, RES_3, RES_7, RES_8, RES_9, RES_12

Tabla 66. Especificación del componente COMP_9

Por último, se incluye una matriz de trazabilidad que relaciona los requisitos de usuario obtenidos en la fase de análisis con los componentes descritos durante la fase de diseño, de esta forma podemos saber de forma rápida y sencilla si todos los requisitos son cubiertos por los componentes del sistema.

COMPONENTE / RU	COMP_1	COMP_2	COMP_3	COMP_4	COMP_5	COMP_6	COMP_7	COMP_8	COMP_9
CAP_1	x								
CAP_2	x								
CAP_3	x								
CAP_4	x								
CAP_5	x								
CAP_6	x								
CAP_7	x								
CAP_8	x								
CAP_9	x								
CAP_10		x							
CAP_11		x							
CAP_12		x							
CAP_13		x							

CAP_14		x							
CAP_15						x			x
CAP_16						x			x
CAP_17					x				
CAP_18					x				
CAP_19			x	x					
CAP_20	x								
CAP_21								x	
CAP_22							x		
CAP_23								x	
CAP_24							x		
CAP_25							x		
CAP_26							x		
CAP_27							X		
CAP_28							x		
CAP_29									x
RES_1	x	x	x	x	x	x	x	x	
RES_2	x	x	x	x	x	x	x	x	
RES_3									x
RES_4						x			
RES_5						x			
RES_6	x	x	x	x	x	x	x	x	
RES_7									x
RES_8									x
RES_9	x	x	x	x	x	x	x	x	x
RES_10	x								
RES_11	x								
RES_12									x

Tabla 67. Matriz de trazabilidad

5.3.2 Diseño de la base de datos

La aplicación desarrollada en realidad no hará uso de una base de datos pues la cantidad de datos que debe almacenar es muy pequeña. En su lugar se van a utilizar archivos XML que serán leídos por un analizador sintáctico y que cargarán estos datos en la aplicación cuando sea necesario, siendo la estructura de estos archivos la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<actualip>192.168.0.192:casa</actualip>
<iplist>
<ip>163.117.87.243:despacho</ip>
<ip>192.168.0.192:casa</ip>
</iplist>
<dir>/sdcard/slides/downloads</dir>
</config>
```

Figura 56: Archivo de configuración

ELEMENTO	DESCRIPCIÓN
<actualip>	Dirección IP del servidor con el que se quiere establecer la conexión
<iplist>	Lista de direcciones IP de servidores conocidos
<ip>	Entrada de la lista de IP que almacena el valor de la dirección y el alias que se le asignó
<dir>	Ruta al directorio en el que se almacenan los archivos recibidos desde el servidor

Tabla 68. Parámetros de configuración

```
<?xml version="1.0" encoding="UTF-8"?>
<transference>
<item name="2010-06-27 22.49.33.jpg" size="431672"
    date="13/10/2010 23:20:55" complete="100" type="0" />
<item name="2010-06-27 22.49.04.jpg" size="427803"
    date="13/10/2010 23:23:56" complete="100" type="0" />
<item name="conf.xml" size="235"
    date="17/10/2010 23:20:13" complete="100" type="0" />
</transference>
```

Figura 57: Archivo de transferencias

ELEMENTO	DESCRIPCIÓN
name	Nombre del archivo transferido.
size	Tamaño en bytes del archivo transferido.
date	Fecha en la que la transferencia tuvo lugar.
complete	Indica si la transferencia se ha completado.
type	Indica el tipo de transferencia, 0 envío y 1 recepción de archivos.

Tabla 69. Información de transferencias

5.3.3Diseño de la interfaz de usuario

En este apartado se van mostrar las distintas interfaces diseñadas para la aplicación, explicando al mismo tiempo cómo el usuario puede interactuar con las mismas y cuál es el resultado de sus acciones.

Al iniciarse la aplicación se muestra una pantalla inicial que inmediatamente pasa a mostrar el explorador de archivos. El explorador tiene un formato de lista en el que cada elemento puede ser un archivo o un directorio.

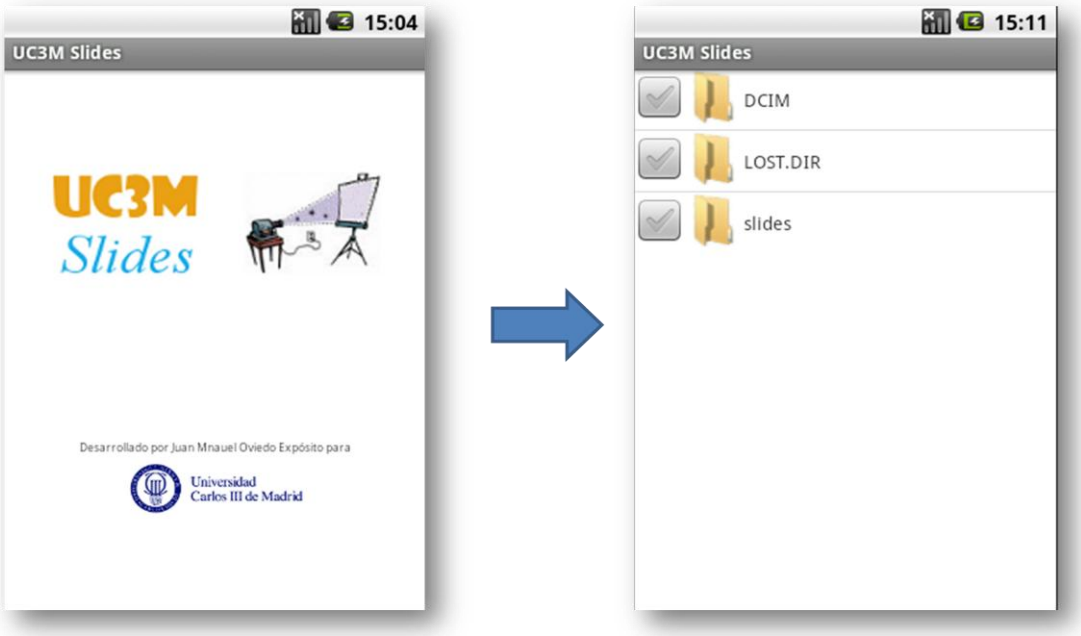

Figura 58: Interfaz inicial

La navegación por directorios se realiza pulsando sobre uno de los elementos del listado de este tipo o pulsando el botón BACK del dispositivo para volver hacia atrás. Si por el contrario seleccionamos un archivo y este puede ser abierto por una aplicación del sistema se mostrará un dialogo en el que se pide elegir una aplicación para abrir el archivo elegido.

Figura 59: Explorar archivos y carpetas

Si se pulsa sobre la casilla de verificación a la izquierda el elemento quedará seleccionado, pudiéndose escoger varios elementos a la vez para posteriormente realizar una de las acciones disponibles.

Otra parte importante de la interfaz del explorador de archivos es el menú, al que se puede acceder pulsando la tecla MENU del dispositivo. Al realizar esta acción se muestran una serie de opciones que permiten realizan diversas tareas como transferir los elementos que estuviesen seleccionados, crear una presentación, etc., las cuales son mostradas a continuación:

Figura 60: Menú de la aplicación

- **Transferencias**: Se pueden transferir archivos hacia el servidor seleccionando el/los elementos pulsando sobre la casilla de verificación y a continuación pulsando MENU y Transferir. Si el servidor es accesible se mostrará el historial de transferencias con los archivos que se acaban de enviar. En caso contrario se mostrará un mensaje al usuario indicándole un posible error con la conexión.

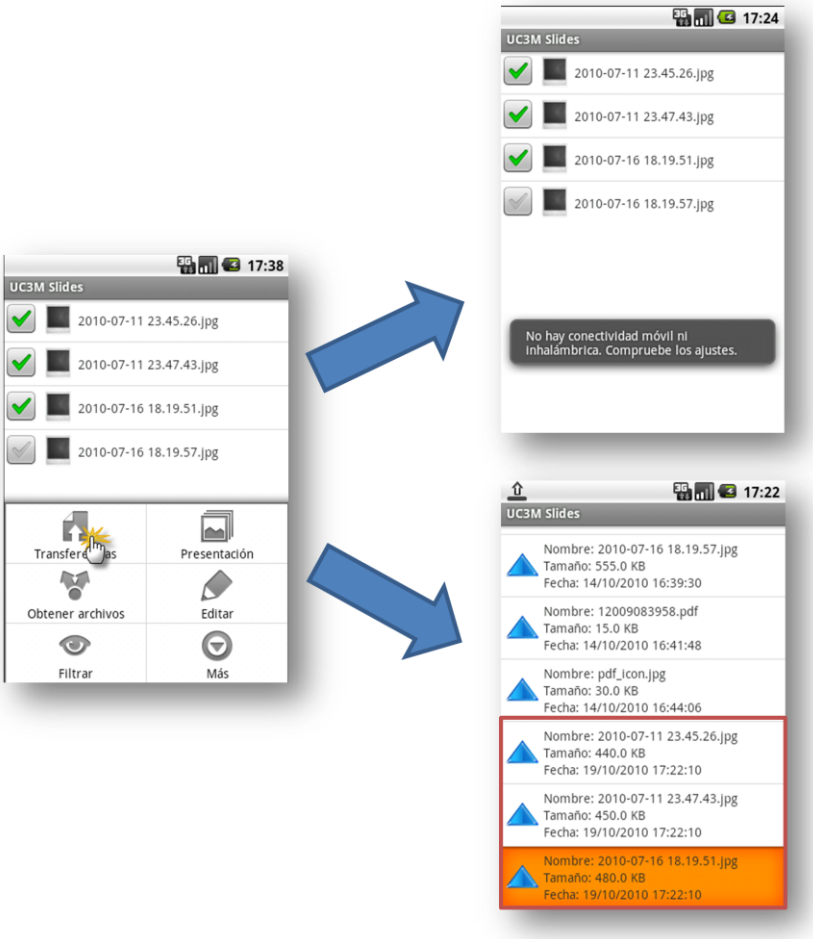

Figura 61: Transferir archivos

- **Presentación:** Esta opción del menú permite controlar las presentaciones de imágenes o documentos PDF y PowerPoint.

  En el caso de una presentación de imágenes, es necesario escoger las mismas antes, y una vez que se han seleccionado se pude pulsar el botón del menú. A continuación se abre un diálogo en el que se pide elegir el tipo de presentación, en este caso seleccionamos la primera opción disponible y en la siguiente interfaz se mostrará una tira con las imágenes seleccionadas.

  Al mismo tiempo en el equipo servidor se abrirá el visor de imágenes predeterminado del sistema pudiendo controlar la presentación con gestos en la pantalla del teléfono.

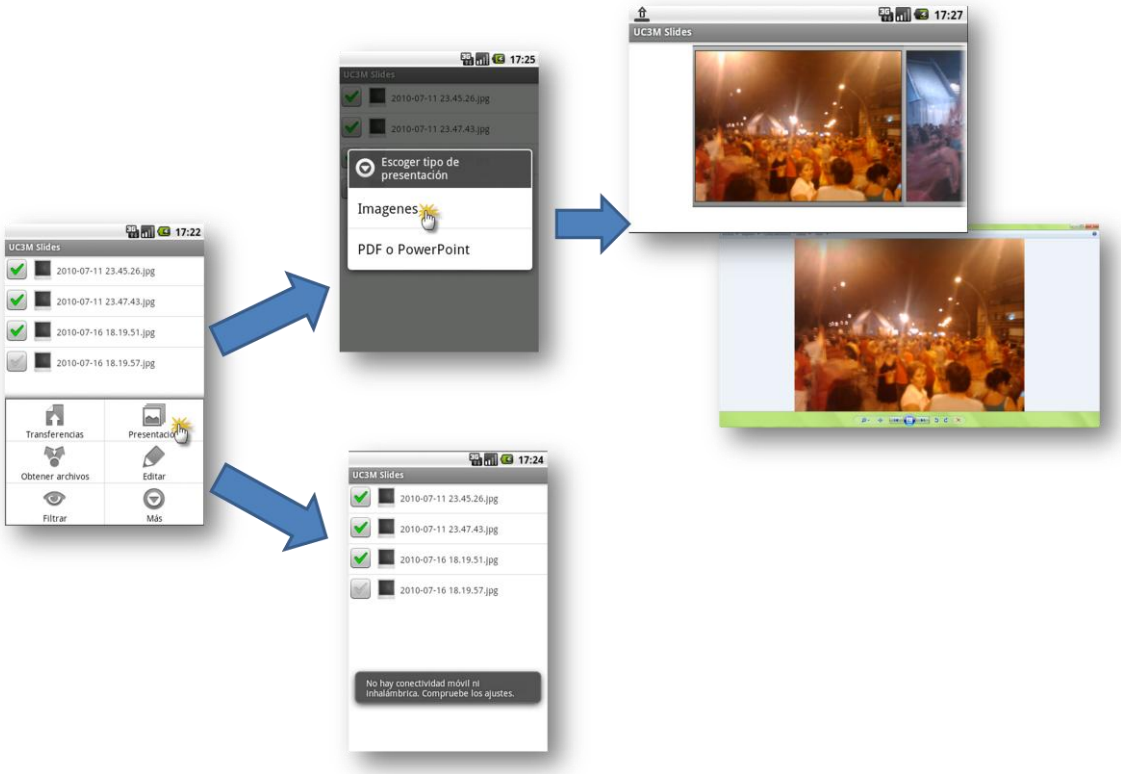

Figura 62: Controlar presentación de imágenes

En el segundo caso, la interfaz mostrará los controles necesarios para interactuar con el servidor como por ejemplo poner en pantalla completa la presentación o avanzar en la misma.

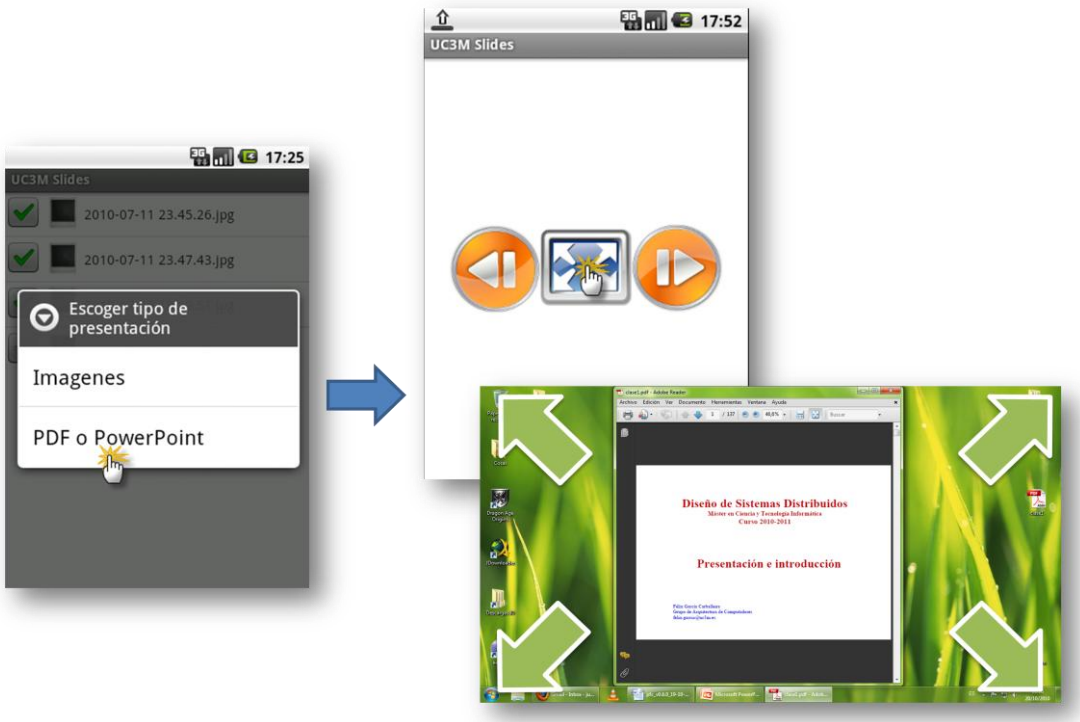

Figura 63: Controlar presentación

Por último si se elije esta opción ene le menú de la aplicación y no se ha elegido un archivo, se pasará a lo que se denominará "modo oyente". Este modo permite a otros usuarios obtener las transparencias que se están proyectando y añadirle anotaciones para poder verlas posteriormente.

Figura 64: Anotaciones

- **Obtener archivos:** La siguiente opción del menú permite descargar archivos desde el servidor.

  Al iniciar esta acción se muestra un listado de los contenidos que son accesibles desde el servidor. Al seleccionar uno de ellos comenzará la descarga de este archivo que será almacenado en el directorio de descargas de la aplicación.

Figura 65: Obtener archivos desde el servidor

- **Editar:** con esta opción del menú accedemos a las acciones básicas de edición de archivos y directorios.

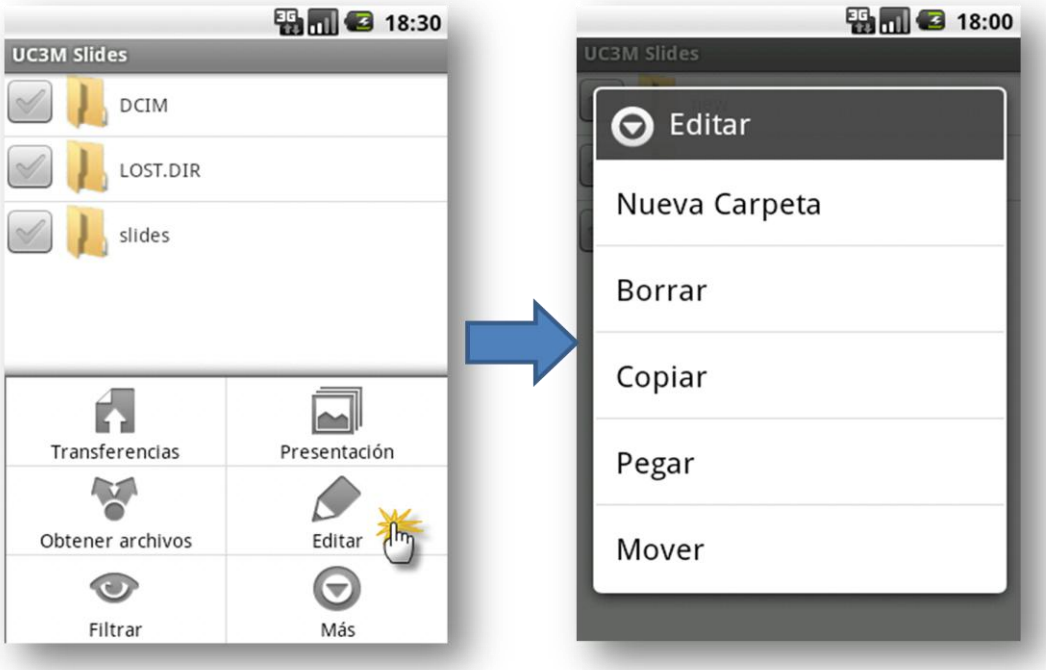

Figura 66: Mostrar menú de edición

- Nueva carpeta: seleccionamos desde editar la opción nueva carpeta. En el diálogo abierto escribir el nombre del nuevo directorio y darle a aceptar.

Figura 67: Crear nueva carpeta

- Borrar: para borrar un elemento (o varios) bastará con seleccionarlos acceder a editar y seleccionar esta opción. Si el elemento a borrar es un directorio se pedirá confirmación para borrar todos los elementos que contenga.

Figura 68: Borrar elemento

- Copiar y pegar: estas acciones son utilizadas con archivos y directorios para realizar copias de los elementos seleccionados. No se permite copiar directorios en ubicaciones que son subdirectorios del mismo.

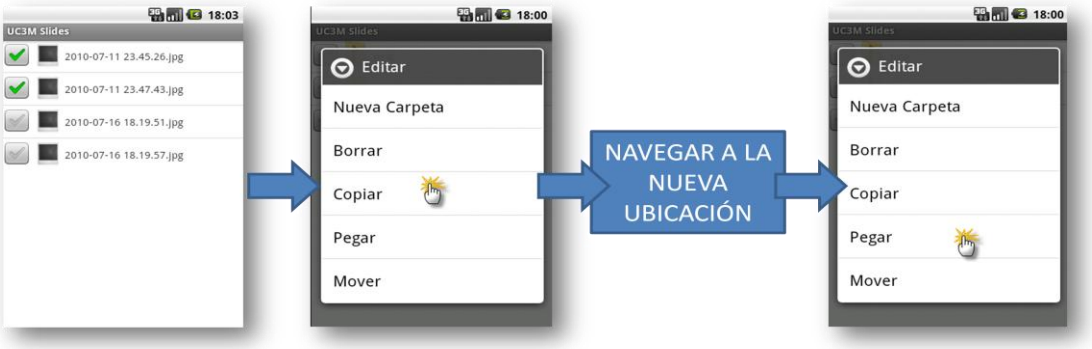

Figura 69: Copiar elementos

- Mover y pegar: estas acciones son utilizadas con archivos y directorios para mover los elementos seleccionados a una nueva ubicación. No se permite mover directorios a ubicaciones que son subdirectorios del mismo.

Figura 70: Mover elementos

A parte de estas acciones accesibles desde el menú de edición es posible realizar un par de acciones extra abriendo el menú de contexto de la actividad. El menú de contexto puede equipararse al menú que se puede ver al pulsar el botón derecho de un ratón sobre un elemento, por ejemplo en el explorador de Windows donde se muestras varias acciones; en este caso se muestran las opciones de renombrar el elemento y ver las propiedades del mismo.

- Renombrar: para abrir el menú de contexto basta con mantener pulsado varios egundos un elemento del listado. En el diálogo que se abre pulsaremos Renombrar para a continuación indicar el nuevo nombre del elemento.

Figura 71: Renombrar elemento

- Propiedades: al abrir el menú de contexto y seleccionar esta opción se abre un nuevo diálogo en el que se pueden ver ciertas propiedades del archivo o directorio.

Figura 72: Ver propiedades

- **Filtrar:** al elegir esta opción desde el menú de la aplicación, se muestra un nuevo diálogo en el que se muestra los distintos filtros disponibles.

  Se pueden aplicar en base al tipo de archivo (imágenes, video, audio, PDF, etc.) y como resultado el usuario obtiene un listado con los archivos que se han encontrado del tipo escogido.

Si el filtro es de imágenes, la interfaz mostrada permitirá ver las mismas como una secuencia. Realizando gestos sobre la pantalla en la zona de la misma donde se muestra la secuencia se podrá cambiar la imagen mostrada en la zona inferior.

Figura 73: Filtro de imágenes

Pulsando en el botón MENU del dispositivo podemos pulsar sobre la opción transferir para enviar al servidor la imagen seleccionada.

El resto de opciones de filtrado muestra un listado de los elementos que han pasado el filtro, pudiendo ser transferidos al servidor o ser abiertos por una aplicación externa del dispositivo.

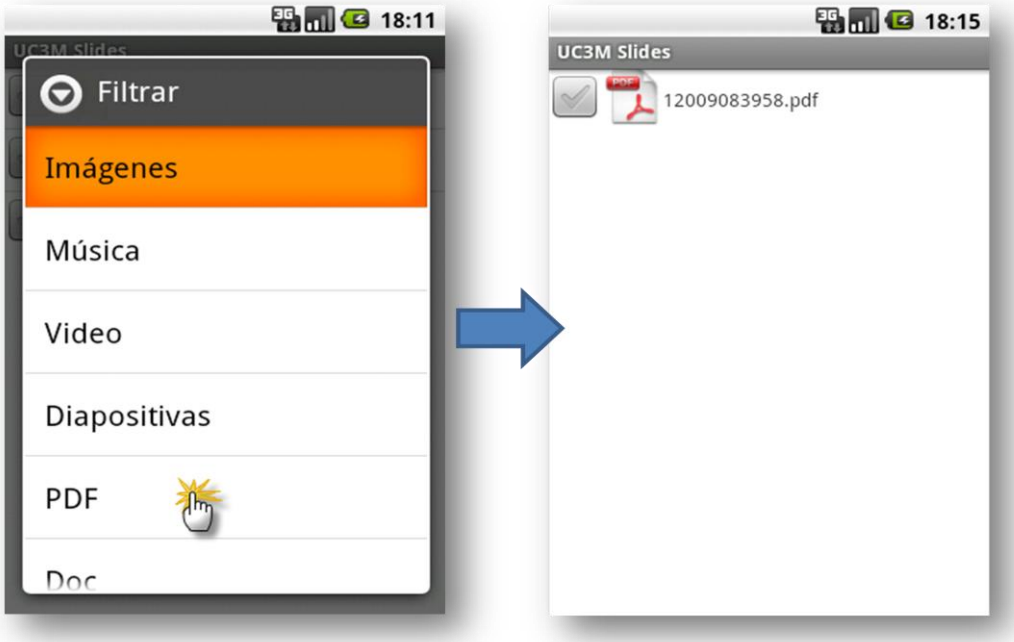

Figura 74: Otros filtros

- **Ajustes:** la siguiente opción del menú de la aplicación permite modificar los ajustes de la misma.

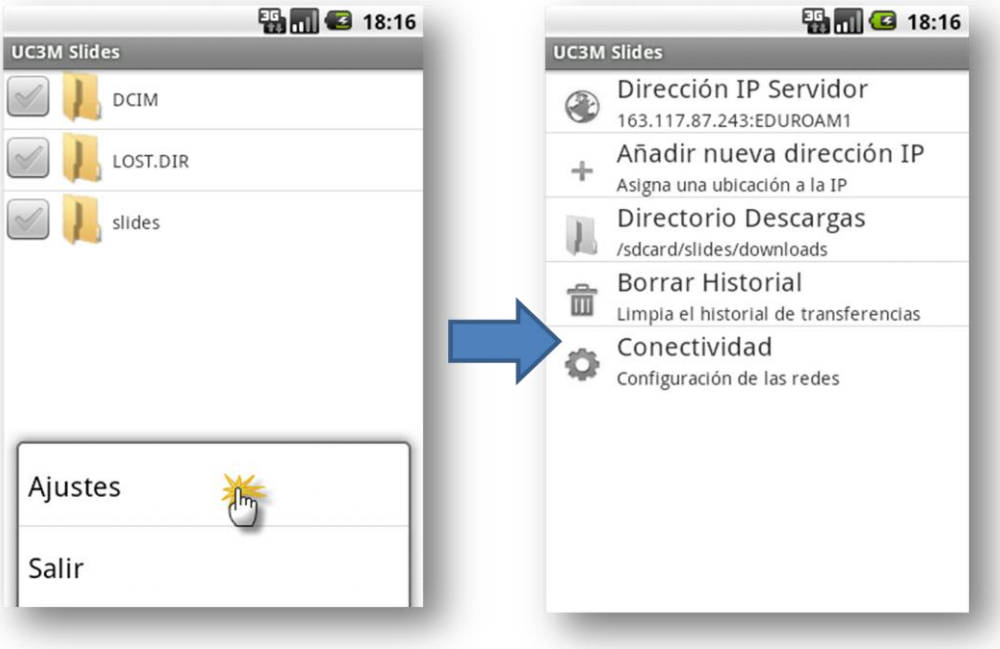

Figura 75: Ajustes

Desde esta vista se pueden cambiar o comprobar los siguientes parámetros:

- Añadir dirección: en el diálogo abierto deberemos introducir la dirección IP del servidor al que queremos conectarnos para interactuar con él y se le puede indicar un alias que permita identificar el servidor de una forma más sencilla.

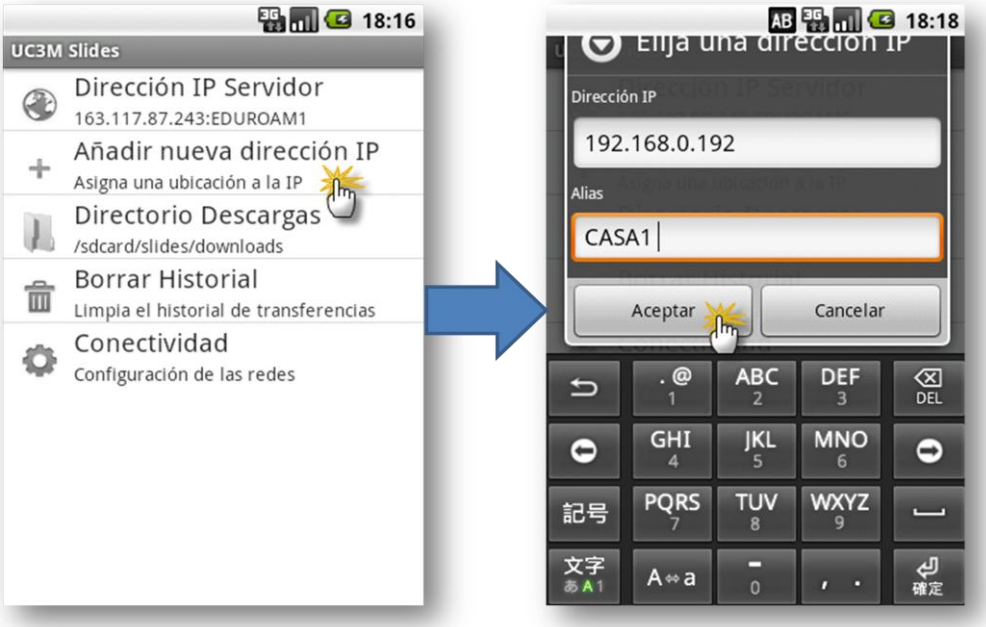

Figura 76: Añadir servidor

- Escoger servidor actual: al pulsar sobre este elemento se abrirá un desplegable en el que se mostrarán las direcciones ya lias guardados en nuestra aplicación. Al elegir una de las direcciones mostradas, automáticamente se emepezará a utilizar esta dirección para realizar las transferencias.

Figura 77: Seleccionar dirección IP del servidor

- Cambiar directorio de descargas: con esta opción se puede modificar el directorio donde se reciben los archivos que provengan del servidor.

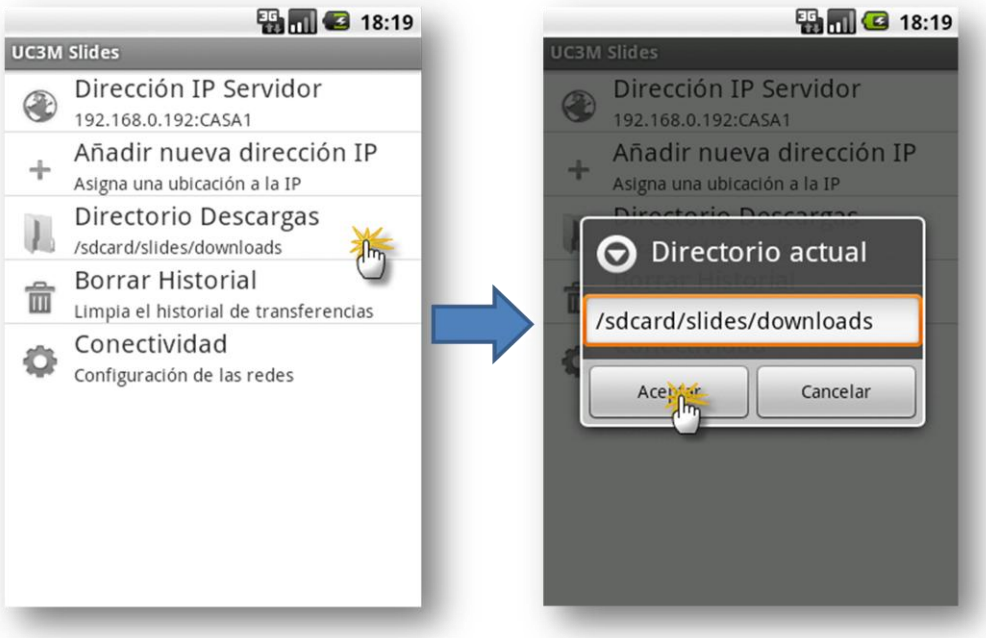

Figura 78: Cambiar directorio de descargas

- Borrar historial de transferencias: la sigueinte opción permite borrar el historial de las tranfernecias realizadas entre dispositivo móvil y servidor.

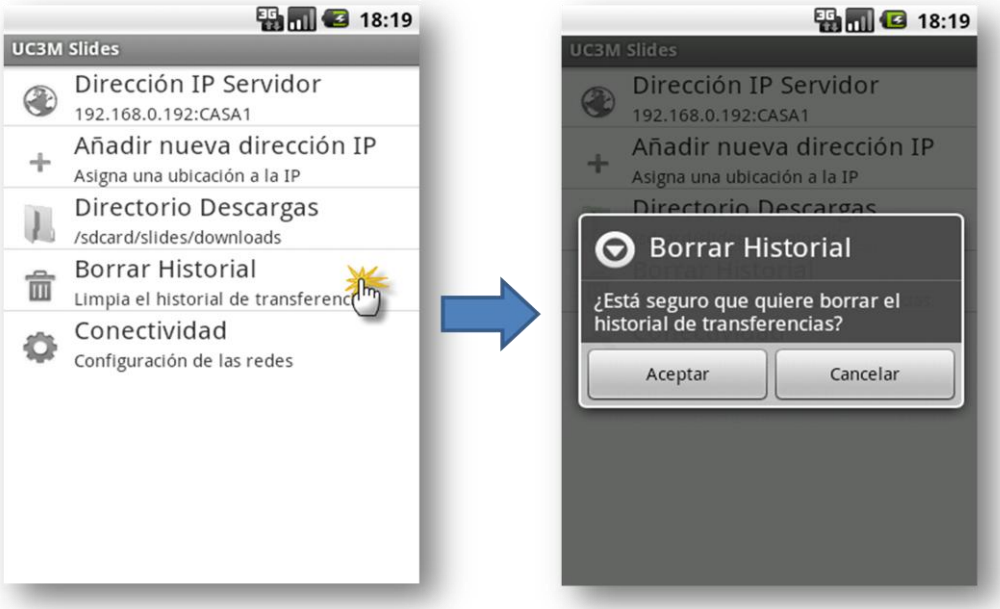

Figura 79: Cambiar directorio de descargas

- Conectividad: esta última opción abre los ajustes del sistema relativos a la conectividad del dispositivo de forma que se puedan cambiar los mismos sin tener que cerrar la aplicación.

Figura 80: Opciones de conectividad del sistema

- **Salir:** habitualmente para salir de una aplicación basta con pulsar el botón BACK en el dispositivo. En lugar de seguir este método se va a indicar al usuario que para finalizar la aplicación debe elegir esta opción desde el menú de la misma tal y como muestra la siguiente figura.

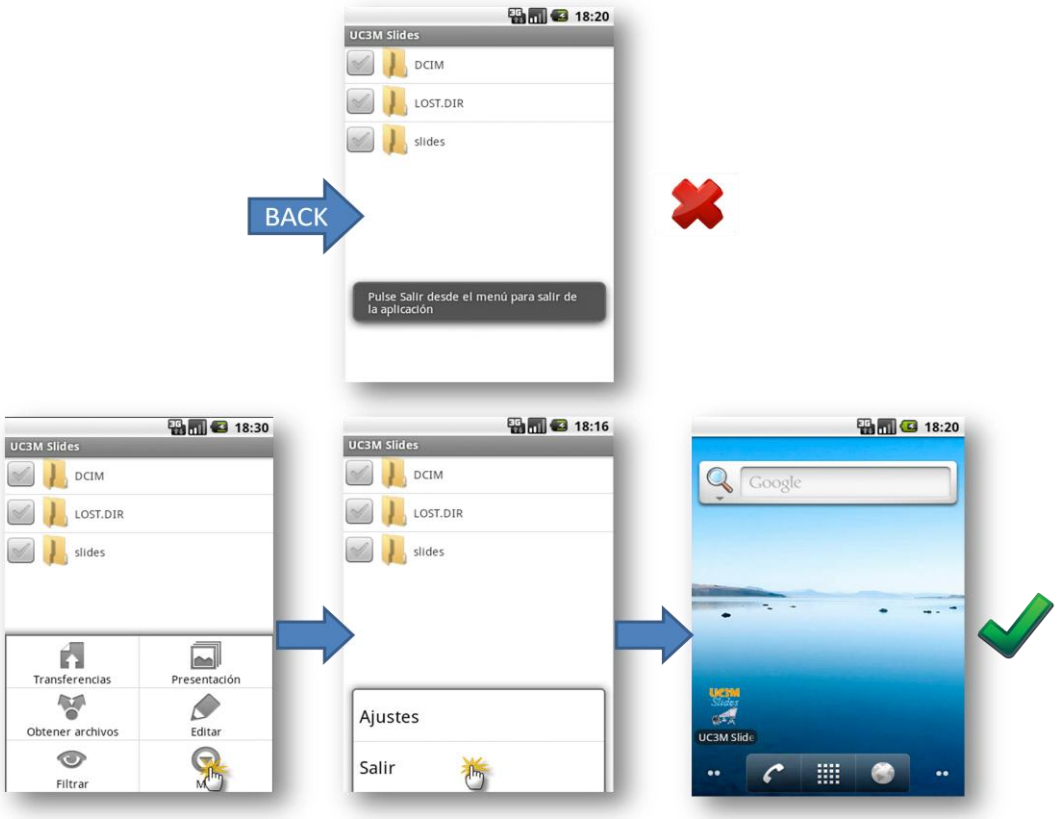

Figura 81: Salir de la aplicación

5.4 Implementación

En esta sección se van a comentar los aspectos de la implementación más importantes y que como desarrollador han supuesto un mayor desafío o han resultado más llamativos o novedosos.

Por tanto, no se pretende que esta sección sea un tutorial de desarrollo si no una forma de exponer los conocimientos o experiencias más importantes que se han adquirido durante el desarrollo del proyecto.

Para otras consideraciones relativas a la implementación se puede consultar el código fuente que se adjunta con este documento.

5.4.1 Explorador de archivos

En primer lugar se va a detallar como se ha implementado este elemento ya que es el componente central de la aplicación desde el que se inician otras actividades de la misma.

Para definir la interfaz de este componente se pensó en mostrar un listado con los elementos que contiene la tarjeta de memoria del teléfono. En Android se cuenta con un *widget* de tipo *ListView* que muestra la información en forma de listado. En el siguiente fragmento de código XML se puede ver como se ha definido el *layout* de esta actividad.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/browser"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" xmlns:android="http://schemas.android.com/apk/res/android">

    <ListView android:id="@id/android:list"
        android:cacheColorHint="#00000000" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <TextView android:id="@id/android:empty" android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:gravity="center"/>
</LinearLayout>
```

Figura 82: Layout del explorador de archivos

El *layout* de esta actividad consta de dos elementos: una lista y un campo de texto. Normalmente, para poder acceder posteriormente a un elemento de un *layout* se le asigna un identificador a dicho elemento, en este caso se le están asignando unos identificadores predefinidos de Android que suelen ser utilizados para definir dos escenarios posibles: si la lista contiene elementos se mostrará el *ListView* y en caso contrario el campo de texto en el que podremos definir un mensaje que informe de esta situación.

Con este archivo XML definimos el *layout* de la actividad pero falta por definir la disposición de los elementos de cada una de las filas en las que estará divida la lista. Para ello se debe crear un nuevo archivo XML tal y como se muestra en la siguiente figura:

```
<?xml  version="1.0"  encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:orientation="horizontal">
    <CheckBox android:id="@+id/check" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:focusable="false" />
    <ImageView android:id="@+id/icon" android:layout_height="45px"
        android:layout_width="45px" />
    <TextView android:id="@+id/rowText" android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>
</LinearLayout>
```

Figura 83: Definición del layout de un elemento de la lista

Con este nuevo archivo se define *layout* lineal con tres elementos: una casilla de verificación, una imagen y un campo de texto situados uno a continuación del otro por el tipo de *layout* elegido.

Una vez que se ha definido la interfaz de usuario de esta actividad el siguiente paso es llenarla de datos. Como se ha comentado, los datos con los que se va a rellenar la interfaz son los archivos y directorios de la tarjeta de memoria del dispositivo. Por tanto, hay que resolver dos problemas: cómo se accede al dispositivo de almacenamiento y como adaptar esta información a un listado.

La primera cuestión es sencilla de resolver pues Android cuenta con métodos para obtener la ruta de acceso al dispositivo de almacenamiento externo como se pude ver en el siguiente fragmento de código:

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    current_path = new Stack();
    current_path.setElementInStack(Environment
            .getExternalStorageDirectory().getAbsolutePath());
    showContents(Environment.getExternalStorageDirectory().listFiles(
            filtro));
    registerForContextMenu(getListView());
} else {
    TextView error = (TextView) findViewById(android.R.id.empty);
    error.setText(R.string.no_sd);
}
```

Figura 84: Acceso a la unidad de almacenamiento externa

A través de la clase *Environment* podemos acceder al estado del dispositivo de almacenamiento externo y si este está montado en el sistema de ficheros del sistema podemos acceder a los contenidos del mismo. En caso contrario podemos asignar un mensaje al campo de texto para informar al usuario que no está insertada la tarjeta de memoria.

Para rellenar la lista con los datos que van a obtenerse del dispositivo de almacenamiento es necesario adaptar estos datos al *ListView*. El término "adaptar" se debe a que para realizar este proceso se debe definir un "adaptador" que a partir de la información que sea almacenada en un arreglos asigne cada posición del mismo a una fila dentro de la lista.

```
class CheckAdapter extends ArrayAdapter<RowModel> {
    Activity context;

    CheckAdapter(Activity context, ArrayList<RowModel> list) {
        super(context, R.layout.row, list);
        this.context = context;
    }
```

Figura 85: Definición de un adaptador

Para conseguir esto, tal y como se ve en la figura anterior, debemos definir una nueva clase que herede del adaptador más adecuado a nuestras necesidades. En este caso es necesario un adaptador de arreglos, al que se le ha asignado un modelo para cada uno de sus elementos. Un modelo no es más que un objeto de Java que podemos definir según nos convenga y que en este caso particular contiene tres elementos: un valor booleano, un entero y una cadena de texto. Si vemos la figura 60 en la que se definía el *layout* de cada fila de la lista, entonces el valor booleano guardara el estado de la casilla de verificación, el entero almacenará el identificador del recurso que se utilizará como imagen y la cadena de texto contendrá la información mostrada por el campo de texto.

Como se puede ver el constructor de esta nueva clase recibe el arreglo que contiene los datos (el listado de ficheros y directorios), y el *layout* que se le va a aplicar a cada uno de ellos indicado por el parámetro "R.layout.row". La clase R es generada de forma automática cada vez que se añade un nuevo recurso al proyecto, ya sean imágenes, archivos de layout o archivos con cadenas de texto. Esta clase sirve para asociar los recursos utilizados en nuestro proyecto con un identificador. De esta manera si añadimos a nuestro proyecto una nueva imagen, bastará con referenciar el identificador que se le ha asignado en la clase R.

En este momento se tiene un objeto que permite adaptar un arreglo a filas de un *ListView,* ¿cómo se muestra el resultado en la interfaz de usuario? Para mostrar cada una de las filas Android hace una llamada al método *getView* que se encarga de generar los objetos *View* de los que se compone la interfaz de usuario como se comentó en el capítulo cuarto. La solución más sencilla para implementar este método es devolver un nuevo objeto *View* en cada llamada, pero resulta ser la solución menos eficiente.

Supongamos un caso en que el listado cuente con pocos elementos, el sistema no se resentirá por generar varios objetos, pero si el listado cuenta con cientos de elementos el rendimiento del terminal se verá afectado por el hecho de tener que gestionar una gran cantidad de memoria. La solución a este problema consiste en reciclar los objetos *View*. Para poder llevar a cabo esta técnica basta con reutilizar el objeto *View* que le llega al método *getView*, tal y como se muestra en el siguiente fragmento de código:

```
public View getView(int position, View convertView, ViewGroup parent) {
    View row = convertView;
    ViewWrapper wrapper;
    CheckBox cb;
    ImageView im;
    TextView tv;

    if (row == null) {
        LayoutInflater inflater = context.getLayoutInflater();

        row = inflater.inflate(R.layout.row, parent, false);
```

Figura 86: Reciclar vistas

Cuando el objeto *convertView* sea nulo, se deberá crear una nueva vista mediante una llamada al servicio del sistema *LayoutInflater* que genera un nuevo objeto *View* a partir del *layout* especificado. Esta solución es óptima para la mayoría de los casos excepto cuando se quiere mantener el estado asociado a un elemento de la lista, como es en este caso con el estado de la casilla de verificación.

Para solventar este problema, y de paso mejorar el rendimiento de la interfaz de usuario, la mejor solución es la comentada en la charla [23], donde se explica cómo aplicar el patrón "*Holder*" de forma que se consigue tener una cache de los objetos creados pudiendo identificar cada objeto con una etiqueta en lugar de utilizando el método *findViewById* del API de Android que tiene un coste computacional más alto.

```
class CheckAdapter extends ArrayAdapter<RowModel> {
    Activity context;

    CheckAdapter(Activity context, ArrayList<RowModel> list) {
        super(context, R.layout.row, list);
        this.context = context;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        View row = convertView;
        ViewWrapper wrapper;
        CheckBox cb;
        ImageView im;
        TextView tv;

        if (row == null) {
            LayoutInflater inflater = context.getLayoutInflater();

            row = inflater.inflate(R.layout.row, parent, false);
            wrapper = new ViewWrapper(row);
            row.setTag(wrapper);
            cb = wrapper.getCheckBox();
            CompoundButton.OnCheckedChangeListener l = new CompoundButton.OnCheckedChangeListener() {
                public void onCheckedChanged(CompoundButton buttonView,
                        boolean isChecked) {
                    Integer myPosition = (Integer) buttonView.getTag();
                    RowModel model = getModel(myPosition);
                    model.isChecked = isChecked;
                }
            };

            cb.setOnCheckedChangeListener(l);

        } else {
            wrapper = (ViewWrapper) row.getTag();
            cb = wrapper.getCheckBox();
        }

        RowModel model = getModel(position);

        cb.setTag(new Integer(position));
        cb.setChecked(model.isChecked);
        tv = wrapper.getTextview();
        tv.setText(model.toString());
        im = wrapper.getImageview();
        im.setImageResource(model.id);
        return (row);
    }
}
```

```
private class ViewWrapper {
    View base;
    CheckBox cb = null;
    ImageView im = null;
    TextView tv = null;

    ViewWrapper(View base) {
        this.base = base;
    }

    CheckBox getCheckBox() {
        if (cb == null) {
            cb = (CheckBox) base.findViewById(R.id.check);
        }
        return (cb);
    }

    TextView getTextview() {
        if (tv == null) {
            tv = (TextView) base.findViewById(R.id.rowText);
        }
        return (tv);
    }

    ImageView getImageview() {
        if (im == null) {
            im = (ImageView) base.findViewById(R.id.icon);
        }
        return (im);
    }
}
```

Figura 87: Código del adaptador de la lista

Tras todas estas optimizaciones se consigue mejorar el rendimiento de la interfaz de usuario de forma notable independientemente del número de elementos de la lista (véase [23]).

5.4.2 Transferencias

Uno de los aspectos más importante de la aplicación son las comunicaciones inalámbricas entre el cliente y el servidor. No solo permiten la transferencia de los archivos en ambos sentidos sino que además son el medio por el que podemos controlar las presentaciones enviando órdenes al equipo servidor.

Al ser un elemento clave para la aplicación se debe buscar una solución que garantice, siempre que sea posible, que la comunicación no va a ser interrumpida. Como es de suponer, si la red 3G o Wifi no está operativa, la aplicación no podrá enviar

datos. Pero ¿qué pasa si el terminal tiene poca memoria disponible y Android determina que la aplicación que tiene que expulsarse es esta?

Como se vio en el tercer capítulo del documento, Android sigue una serie de reglas para liberar los recursos cuando los necesita para otras actividades. Si realizamos las transferencias utilizando hilos en una actividad, se corre el riesgo de que si se cambia a otra tarea y el sistema necesita recursos, esta actividad sea expulsada y por tanto las transferencias sean canceladas.

Para evitar esto, se ha implementado un servicio en lugar de una actividad para que realice las transferencias de forma transparente al usuario.

```
@Override
public void onCreate() {
    cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
    mNM = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    is_alive = false;
    ip_set = false;
    Timer connect = new Timer();
    connect.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            if (!ip_set)
                loadConfiguration();
            if (ip_set) {
                NetworkInfo ni_mobile = cm
                        .getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
                NetworkInfo ni_wifi = cm
                        .getNetworkInfo(ConnectivityManager.TYPE_WIFI);
                if (ni_mobile.isConnected() || ni_wifi.isConnected()) {
                    if (!is_alive) {
                        initReceiverConnection();
                    }
                } else
                    is_alive = false;
            }
        }
    }, 0, 10000);
}
```

Figura 88: Creación del servicio

De esta manera, el proceso que se encuentre ejecutando la aplicación estará asignado a la categoría o nivel de “Procesos en primer plano”, y como se comentó en el apartado 3.4 de este documento, estos procesos serán los últimos en ser expulsados de la memoria cuando se tenga que liberar recursos.

5.4.3 Control de presentaciones

Dada la intención multiplataforma del proyecto, al iniciarse el mismo, uno de los grandes problemas que se encontraron fue cómo podría ejecutarse una aplicación que estuviese asociada a un tipo de archivo de forma independiente a la plataforma.

Una primera idea fue lanzar procesos nativos del sistema operativo anfitrión a través de Java. SI bien esta idea podía llevarse a cabo, existía una solución aún más sencilla y que ofrecía los mismos resultados.

Esta solución es la clase Desktop de Java (disponible desde la versión del JRE 1.6) que permite, entre otras cosas, abrir la aplicación asociada con un archivo. Por tanto con muy pocas líneas de código este problema queda resuelto tal y como puede verse en la siguiente figura:

```
public class AppLauncher {

    private Desktop d;

    public AppLauncher() {
        d = Desktop.getDesktop();
    }

    public boolean launchApp(String path_to_file) {
        try {
            d.open(new File(path_to_file));
            return true;
        } catch (IOException e) {
            return false;
        }
    }
}
```

Figura 89: Lanzar aplicación

El siguiente reto, después de conseguir abrir la aplicación, es poder controlarla de forma remota traduciendo las acciones que se realicen sobre la pantalla del dispositivo móvil en otras que sean entendidas por la aplicación que necesitemos. Gracias a la clase Robot de Java, este objetivo se consigue de una forma muy sencilla.

Esta clase incluye métodos que controlan el movimiento del ratón, el teclado o que permiten realizar capturas de la pantalla. Por tanto, asociando una acción en el dispositivo con una combinación de teclas se consigue poder realizar acciones como pasar a otra diapositiva o activar el modo de pantalla completa.

```
case 0: // forward
    robot.keyPress(KeyEvent.VK_RIGHT);
    robot.keyRelease(KeyEvent.VK_RIGHT);
    break;
case 1: // backward
    robot.keyPress(KeyEvent.VK_LEFT);
    robot.keyRelease(KeyEvent.VK_LEFT);
    break;
case 2: // enter full screen
    robot.keyPress(KeyEvent.VK_F11);
    robot.keyRelease(KeyEvent.VK_F11);
    break;
case 3: // exit full screen
    robot.keyPress(KeyEvent.VK_ESCAPE);
    robot.keyRelease(KeyEvent.VK_ESCAPE);
    break;
case 4: // pdf full screen
    robot.keyPress(KeyEvent.VK_CONTROL);
    robot.keyPress(KeyEvent.VK_L);
    robot.keyRelease(KeyEvent.VK_L);
    robot.keyRelease(KeyEvent.VK_CONTROL);
    break;
case 5: // ppt full screen
    robot.keyPress(KeyEvent.VK_F5);
    robot.keyRelease(KeyEvent.VK_F5);
    break;
```

Figura 90: Mapear acciones con teclas

5.4.4Localización

Una de las características de Android que resulta ser bastante útil es la forma en la que el sistema utiliza los recursos que tiene disponibles. En el tercer capítulo del documento se hablaba de cómo definir recursos alternativos en función de ciertos parámetros del dispositivo.

Uno de estos parámetros es la localización de los menús y textos mostrados en la interfaz de usuario configurable desde el menú de ajustes del dispositivo.

Figura 91: Seleccionar idioma

La mejor forma de traducir una aplicación a varios idiomas distintos es definir recursos en función de la localización o idioma del dispositivo. En este caso se deben definir varios recursos para el archivo "strings.xml" que es de donde la aplicación obtiene habitualmente las cadenas de texto que son mostradas en la interfaz de usuario. Para ello bastará con definir dentro del directorio "res" del proyecto dos subdirectorios "*values*" y "*values_es*", el primero será utilizado por defecto y el segundo será utilizado cuando el idioma del dispositivo sea el español.

Figura 92: Interfaz localizada

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

Cuando se inició este proyecto se plantearon una serie de objetivos genéricos ligados al prototipo que se quería desarrollar. Sin embargo el propio proceso de desarrollo también tenía sus propios objetivos, siendo el primer objetivo fundamental el de conocer la plataforma de Android, puesto que inicialmente no tenía conocimiento alguno de la misma.

La búsqueda de una aplicación que fuera suficientemente útil para ser usada, abordable por una persona en un plazo de tiempo razonable y variada como para aprender los aspectos típicos de esta plataforma no fue en absoluto trivial, pero fue un reto que me permitió no solo poner en práctica la mayoría de conocimientos aprendidos durante mis estudios, sino también mejorar mi capacidad de auto-aprendizaje.

Para empezar la búsqueda de una aplicación que fuera útil me basé en la siguiente pregunta: *¿Qué necesitas o hechas en falta?*

Dándole vueltas a esta pregunta surge la idea de realizar una aplicación que gestione los contenidos de la tarjeta de memoria y que además pueda transferir los mismos entre el teléfono y el ordenador, todo ello sin cables. Pero no contento con esto, ¿por qué no añadirle control remoto de ciertas aplicaciones?

Al final esta última idea resultó ser el núcleo de todo este proceso de desarrollo que se ha explicado a lo largo del documento, abarcando desde el análisis de los requisitos de la aplicación hasta la implementación de la misma.

Por tanto el objetivo principal del proyecto se ha visto satisfecho ampliamente pues se ha conseguido desarrollar una aplicación que permite realizar presentaciones utilizando para ello un dispositivo móvil, permitiendo además a otras personas poder ser participes de la presentación pudiendo obtener las diapositivas mostradas para añadir sus anotaciones personales.

En el proceso para satisfacer este objetivo se han visto cumplidos igualmente los demás objetivos que fueron planteados al comienzo del proyecto. Pese a que la idea inicial fue desarrollarlo en Android, también se estudiaron las alternativas existentes como iOS, Symbian y Windows Phone, pero por los motivos explicados al comienzo de este documento se siguió con el desarrollo en la plataforma de Google.

Es evidente que, fruto de todo este trabajo, se ha conseguido satisfacer también los objetivos planteados de conocer los detalles necesarios de la plataforma y establecer los pasos necesarios para desarrollar aplicaciones para ella, pues en caso contrario no estaría escribiendo estas líneas.

Desde un punto de vista personal, la realización de este proyecto me ha servido para descubrir un gran interés por el mundo del desarrollo en el campo de la telefonía móvil. Si tenemos en cuenta la situación actual de este mercado, donde se venden millones de terminales al año y cientos de millones de aplicaciones, resulta que tengo ante mí una muy buena oportunidad de futuro.

En conclusión, la realización de este proyecto ha servido para satisfacer todos los objetivos planteados al inicio del mismo, a la vez que ha abierto una puerta hacia un futuro profesional que al comienzo del mismo no existía.

6.2 Líneas futuras

Tras la finalización del proyecto han surgido algunas líneas de trabajo bastante interesantes y que sería oportuno evaluar su realización.

- Seguridad:
  - Estudiar el impacto en el rendimiento de la aplicación que tendría el envío de datos a través de canales más seguros, utilizando para ello mecanismos como el cifrado de datos (Socket SSL).
  - Establecer un mecanismo seguro de acceso a las anotaciones personales.

- Aspecto colaborativo:
  - Aumentar este aspecto de la aplicación permitiendo realizar presentaciones a varias personas a la vez o permitiendo a los espectadores interactuar de algún modo con los contenidos de la presentación como por ejemplo realizando encuestas.
  - Realizar itinerarios de las presentaciones en función de los conocimientos expresados por la audiencia de la presentación.
- Comunicaciones:
  - Añadir soporte para comunicación bluetooth que asegure que si la conexión de la red local o de Internet está experimentando algún problema de conectividad se pueda seguir realizando la presentación de forma normal.
  - Añadir geo-localización para que la aplicación se auto-configure conectándose automáticamente a un servidor conocido en función de lo cerca que este de este.
- Funcionalidad:
  - Integrar el control del ratón a través de la pantalla del dispositivo, expandiendo de forma notable las capacidades de control remoto de la aplicación.

Capítulo 7

Planificación y presupuesto

7.1 Planificación

A continuación se va a detallar la planificación seguida para la elaboración del proyecto. Se ha de tener en cuenta que la dedicación a este proyecto no ha sido exclusiva, pues de forma simultánea se han desarrollado otras actividades que no han permitido dedicar todos los esfuerzos al mismo.

En la siguiente tabla se pueden ver las actividades realizadas durante la elaboración del proyecto indicándose la fecha de inicio y finalización de cada una de ellas así como la duración en días de cada actividad.

Actividad	Fecha inicio	Duración (días)	Fecha fin
Estudio de viabilidad	20/10/2009	10	30/10/2009
Documentación inicial	01/11/2009	75	14/01/2010
Análisis	15/01/2010	32	14/02/2010
Diseño	15/02/2010	59	14/04/2010
Implementación	15/04/2010	168	30/09/2010
Documentación del proyecto	20/06/2010	124	22/10/2010
Pruebas	15/09/2010	35	20/10/2010

Tabla 70. Planificación

La siguiente figura muestra el diagrama de Gantt asociados a la planificación descrita en la tabla anterior:

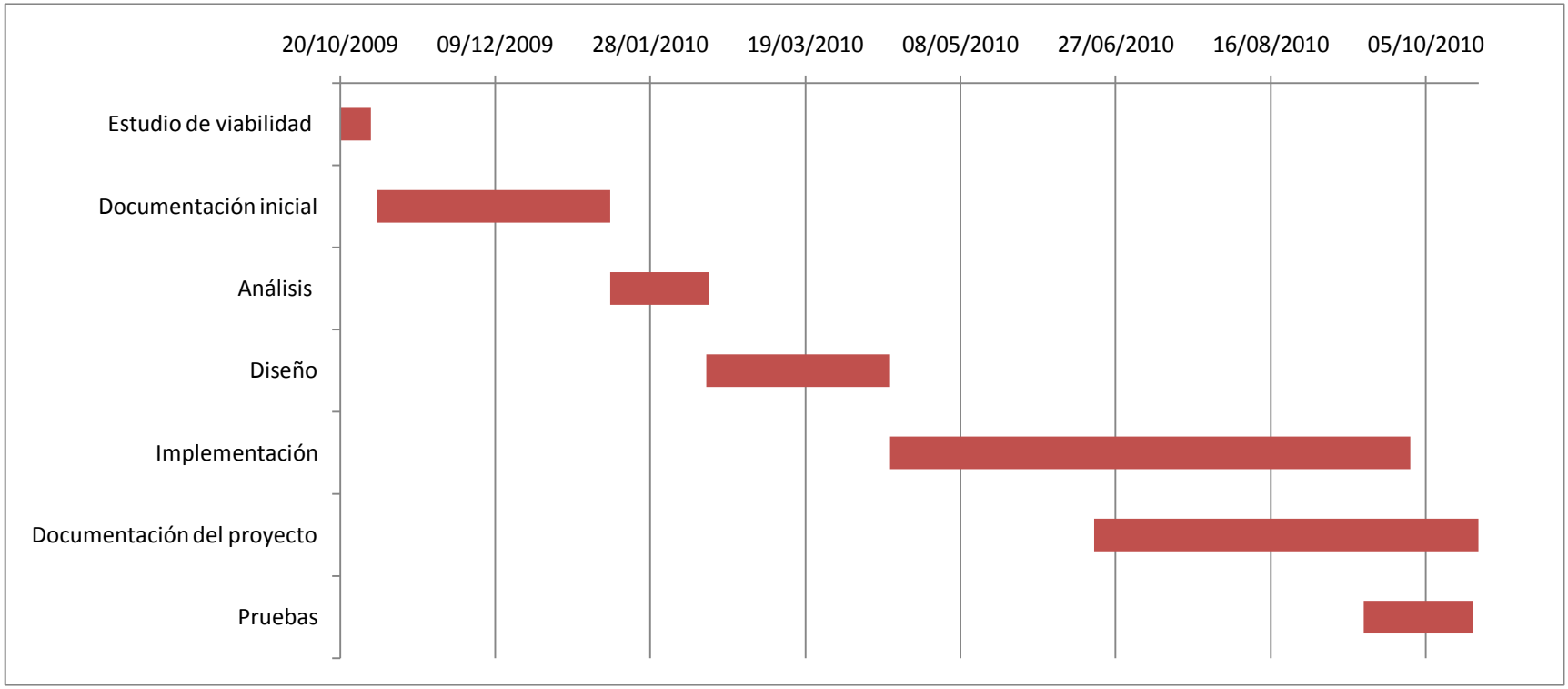

Figura 93: Diagrama Gantt para la planificación del proyecto

7.2 Presupuesto

En esta sección se detallará el presupuesto estimado para el proyecto, detallando los gastos de personal, equipo y otros gastos siguiendo las guías propuestas por la Universidad Carlos III de Madrid [22].

Los gastos de personal se han calculado teniendo en cuenta los siguientes aspectos:

- Se han dedicado tres horas por día al proyecto
- No se contabilizan los siguientes días festivos:
  - 7, 8, 24, 25 y 31 de Diciembre de 2009
  - 1 y 6 de Enero de 2010
  - 19 de Marzo de 2010
  - 1 y 2 de Abril de 2010
  - 3 de Junio de 2010
  - 12 de Octubre de 2010
- No se contabiliza el mes de Agosto de 2010

Teniendo en cuenta estas consideraciones y la planificación del apartado anterior, los días de dedicación real al proyecto para cada actividad se muestran en la siguiente tabla:

Actividad	Días reales
Estudio de viabilidad	9,00
Documentación inicial	52,00
Análisis	21,00
Diseño	40,00
Implementación	98,00
Documentación del proyecto	67,00
Pruebas	25,00

Tabla 71. Días dedicados al proyecto

Conociendo estos datos se pueden obtener los gastos de personal asociados al proyecto de la siguiente forma:

Coste de personal = ((Total días x Horas/día) / Dedicación hombre/mes) * Coste hombre/mes

Total días = 936 días

Horas/día = 3

Dedicación hombre mes = 131.25 horas

Coste hombre mes = 2694.38 €

Como resultado los gastos de personal se elevan a DIECINUEVE MIL DOSCIENTOS SESENTA Y CUATRO EUROS CON OCHENTA Y CUATRO CÉNTIMOS DE EURO.

Para la realización del proyecto se han adquirido además varios equipos, con un coste que queda detallado a continuación:

- Ordenador portátil HP dv6 1278ss con un coste de 768 €
- Teléfono HTC Magic con un coste de 189 €

No obstante siguiendo las consideraciones expuestas en [22] se debe calcular la amortización de estos equipos en el proyecto tal y como se indica a continuación:

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable *
Portátil	768,00	100	12	60	153,60
Teléfono HTC Magic	189,00	100	12	60	37,80
				Total	191,40

Tabla 72. Amortización equipos

*) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} xCxD$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto

Por último, se incluye además una relación de gastos imputables al proyecto que son los siguientes:

- Licencia de desarrollo de Android con un coste de 25 €
- Licencia de Microsoft Office 2007 Professional (descarga) con un coste de 109.90 €
- Licencia de Microsoft Project 2007 Professional (descarga) con un coste de 129 €
- Licencia de Microsoft Visio 2007 Professional (descarga) con un coste de 119.90 €
- Tarifa de datos 3g con un coste total de 180 € (15 € x 6 meses)

De esta forma los costes de funcionamiento del proyecto ascienden a QUINIENTOS SESENTA Y TRES EUROS CON OCEHNTA CÉNTIMOS.

En resumen, el presupuesto final del proyecto quedaría configurado como se muestra a continuación:

CONCPETO	IMPORTE
Personal	19.265 €
Amortización	191 €
Costes de funcionamiento	564 €
Costes Indirectos (20%)	4.004 €
Total (Sin IVA)	24.024 €
TOTAL (IVA 18%)	28.348,32 €

Tabla 73. Presupuesto final

El presupuesto total de este proyecto asciende a la cantidad de VEINTIOCHO MIL TRESCIENTOS CUARENTA Y OCHO EUROS CON TREINTA Y DOS CÉNTIMOS DE EURO.

La cantidad invertida podría verse recuperada vendiendo la aplicación en Android Market o añadiendo publicidad a la misma con el programa AdSense de Google. Respecto a la publicidad, no se tienen datos exactos de cuánto se puede obtener por este medio ya que el dinero ganado depende de mucho factores. La opción de vender la aplicación por ejemplo a un precio de dos euros, de los que Google obtiene el treinta por ciento; supone que se deberían vender cerca de diecinueve mil quinientas aplicaciones para recuperar la inversión inicial.

Leganés a 29 de Octubre de 2010

El ingeniero proyectista

Fdo. Juan Manuel Oviedo Expósito

Glosario

ADT	*Android Developers Tools*
A-GPS	*Assisted GPS o Sistema de geo-posicionamiento asistido*
APK	*Android Package*
A2DP	*Advanced Audio Distribution Profile, define cómo se puede propagar un stream de audio (mono o estéreo) entre dispositivos a través de una conexión Bluetooth*
AVRCP	*Audio/Video Remote Control Profile o Perfil de control remoto de audio/vídeo.*
EUA	*Estado Unidos de América*
Framework	*Estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado*
GHZ	*Gigahercio*
GPS	*Global Positioning System o Sistema de Posicionamiento Global*
IDE	*Integrated Development Environment*
IVA	*Impuesto al valor añadido*
JAR	*Java Archive*
JIT	*Just in Time compilation*

JVM	*Java Virtual Machine*
MB	*Mega Byte*
PDA	*Personal Digital Assistant*
QWERTY	*Disposición de las teclas del teclado, siendo QWERTY la más común*
SMS	*Short Message Service*
SDK	*Software Development Kit*
URI	*Uniform Resource Identifier*
WXGA	*Wide eXtended Graphics Array es una norma de visualización de gráficos de ordenador para una resolución de 1366x768 pixeles*
XML	*Extended Markup Language*

Referencias

[1] Definición de dispositivo móvil: http://es.wikipedia.org/wiki/Dispositivo_m%C3%B3vil
[2] Datos del mercado de telecomunicaciones en España a Marzo de 2009: http://www.cmt.es/es/publicaciones/anexos/NM_marzo2009.pdf
[3] Estimación número de líneas móviles a nivel mundial: http://communities-dominate.blogs.com/brands/2009/02/bigger-than-tv-bigger-than-the-internet-understand-mobile-of-4-billion-users.html
[4] Definición de smartphone: http://es.wikipedia.org/wiki/Smartphone
[5] Página de desarrollo de Android: ¿Qué es Android?: http://developer.android.com/guide/basics/what-is-android.html
[6] Acerca de Android: http://es.wikipedia.org/wiki/Android
[7] Acerca de Android (Inglés): http://en.wikipedia.org/wiki/Android_%28operating_system%29
[8] Comparativa plataformas iOS, Android y Windows Phone: http://blog.neuronaltraining.net/?p=15221
[9] Symbian: http://www.symbian.org/
[10] Estudio Gartner evolución del mercado: http://www.gartner.com/it/page.jsp?id=1434613
[11] Estudio cuota de mercado plataformas móviles: http://news.cnet.com/8301-1035_3-20016647-94.html
[12] IDE Eclipse: http://www.eclipse.org/downloads/

[13] Android SDK: http://developer.android.com/sdk/index.html
[14] Eclipse ADT Plug-in: https://dl-ssl.google.com/android/eclipse/
[15] ComScore: http://www.comscore.com/
[16] Herramientas de depuración: http://developer.android.com/guide/developing/debug-tasks.html
[17] GMote: http://www.gmote.org/
[18] PPT Remote: http://www.pptremotecontrol.com/index.html
[19] Teclado QWERTY: http://es.wikipedia.org/wiki/Teclado_QWERTY
[20] App Store: http://www.apple.com/es/iphone/apps-for-iphone/#heroOverview
[21] Android Market: http://www.android.com/market
[22] Realización de presupuesto según Universidad Carlos III de Madrid: https://www.uc3m.es/portal/page/portal/administracion_campus_leganes_est_cg/proyecto_fin_carrera/Formulario_PresupuestoPFC-TFG%20(3)_2.xlsx
[23] Google IO 09: Mejorar la interfaz de usuario de las aplicaciones: http://www.google.com/events/io/2009/sessions/TurboChargeUiAndroidFast.html